NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

VHDL MODELING AND SIMULATION OF A DIGITAL IMAGE SYNTHESIZER FOR COUNTERING ISAR

by

Özkan Kantemir

June 2003

Thesis Advisor: Douglas J.Fouts
Second Reader: Phillip E.Pace

Approved for public release, distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** June 2003	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE: VHDL Modeling and Simulation of a Digital Image Synthesizer for Countering ISAR	**5. FUNDING NUMBERS**
6. AUTHOR Özkan Kantemir	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) Office of Naval Research Code 313 Arlington, Virginia	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release, distribution is unlimited	**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(maximum 200 words)***

This thesis discusses VHDL modeling and simulation of a full custom Application Specific Integrated Circuit (ASIC) for a Digital Image Synthesizer (DIS). The DIS synthesizes the characteristic echo signature of a pre-selected target. It is mainly used against Inverse Synthetic Aperture Radars as an electronic counter measure. The VHDL description of the DIS architecture was exported from Tanner S-Edit, modified, and simulated in Aldec Active HDL^{TM}. Simulation results were compared with C++ and Matlab simulation results for verification. Main subcomponents, a single Range Bin Processor (RBP), a cascade of 4 RBP s and a cascade of 16 RBP s were tested and verified. The overhead control circuitry, including Self Test Circuitry and Phase Extractor, was tested separately. Finally overall DIS was tested and verified using the control circuitry and a cascade of 4 RBP s together, representing the actual 512 RBP s. As a result of this research, the majority of the DIS was functionally tested and verified.

14. SUBJECT TERMS Digital Image Synthesizer, DIS, VLSI, ASIC, CMOS, VHDL, Active HDL^{TM}, Aldec, Tanner	**15. NUMBER OF PAGES** 166
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release, distribution is unlimited

VHDL MODELING AND SIMULATION OF A DIGITAL IMAGE SYNTHESIZER FOR COUNTERING ISAR

Özkan Kantemir
First Lieutenant, Turkish Army
B.S., Turkish Army Academy, 1998

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 2003

Author: Özkan Kantemir

Approved by: Douglas J. Fouts
Thesis Advisor

Phillip E. Pace
Second Reader/Co-Advisor

John P. Powers
Chairman, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis discusses VHDL modeling and simulation of a full custom Application Specific Integrated Circuit (ASIC) for a Digital Image Synthesizer (DIS). The DIS synthesizes the characteristic echo signature of a pre-selected target. It is mainly used against Inverse Synthetic Aperture Radars as an electronic counter measure. The VHDL description of the DIS architecture was exported from Tanner S-Edit, modified, and simulated in Aldec Active HDL^{TM}. Simulation results were compared with C++ and Matlab simulation results for verification. Main subcomponents, a single Range Bin Processor (RBP), a cascade of 4 RBP s and a cascade of 16 RBP s were tested and verified. The overhead control circuitry, including Self Test Circuitry and Phase Extractor, was tested separately. Finally, the overall DIS was tested and verified using the control circuitry and a cascade of 4 RBP s together, representing the actual 512 RBP s. As a result of this research, the majority of the DIS was functionally tested and verified.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank to my mother, my father, Seyhan and Serkan for supporting me even from overseas, from Turkey. I tried to be worthy of their love. I also worked hard not to disappoint the people of Turkey whom I have been serving with honor.

I would also like to thank Professor Phillip E. Pace for his support and friendship. I would like to single out Professor Douglas J. Fouts who guided me through the first stumbling steps in implementing VHDL and patiently explained the hardware functionality of our project, professor, you can fill the Pacific with your patience, thanks a lot.

Danielle & Michael Smith, Dennis Evans Sensei, Auntie Kathy, Mitch and Sonya and the Aikido of Monterey family, thank you for your support.

Hakan, Ali, Coskun, Kabalar and Bunyamin, it has been a pleasure to be altogether. Thank you.

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

Synthetic aperture radars (SARs) and inverse synthetic aperture radars (ISARs) are capable of generating images of target objects even under adverse conditions when other sensors are blind. With SAR and ISAR, the ability to detect and identify a contact is greatly improved. Current electronic attack systems (such as decoys and jamming) fail to counter the identification and targeting.

The Digital Image Synthesizer (DIS) is designed to perform this task. If the target platform is able to receive, modify and re-transmit the actual radar signal sent by the ISAR/SAR, the targeting platform would not be able to distinguish between the transmitted signal and the actual radar returns echoed from the target. To do so, the signal intercepted by the target platform must be carefully and precisely manipulated in phase and amplitude such that the deception is not noticeable.

Either digital or analog methods may be used to synthesize a false target radar image. The analog methods are bulky, susceptible to noise and have limited bandwidth, which makes them impractical. A digital method has many advantages over an analog method. The major advantages are its increased bandwidth capacity and its ability to delay signals as long as necessary for a given application. With such a digital method, it is possible for a small ship to appear as large as an aircraft carrier or any high value target.

This thesis discusses modeling and functional verification of the DIS. The VHDL description of the DIS architecture was exported from Tanner S-Edit, modified, and simulated in Aldec Active HDL^{TM}. Modifications to the VHDL source code included renaming of components to comply with VHDL naming conventions and adding behavioral descriptions for some components. Simulation results were compared with C++ and Matlab simulation results for verification. Main subcomponents, a single Range Bin Processor (RBP), a cascade of 4 RBP s and a cascade of 16 RBP s were tested and verified. The overhead control circuitry, including Self Test Circuitry and Phase Extractor, was tested separately. Finally, the overall DIS was tested and verified using the control circuitry and a cascade of 4 RBP s together, representing the actual 512 RBP s. As a result of this research, the majority of the DIS was functionally tested and verified.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION TO DIGITAL IMAGE SYNTHESIZER (DIS)

A. BACKGROUND

The Digital Image Synthesizer (DIS) is an Application Specific Integrated Circuit (ASIC) able to generate false target images to deceive an Inverse Synthetic Aperture Radar (ISAR).

1. Inverse Synthetic Aperture Radar (ISAR)

As explained in detail in [1] and [2], ISAR is a high-resolution radar technique that can develop a two-dimensional intensity image of moving targets in the range and cross-range (Doppler) domains. ISAR imaging is used in many military applications such as target classification, recognition and identification. Surveillance systems such as the U.S. Navy AN/APS-137 ISAR and the Russian Sea Dragon maritime patrol radar use an ISAR 2-D imaging mode to provide detection, classification and tracking capability against surface and surfaced submarine targets.

Figures 1 and 2 (courtesy of the Tactical Electronic Warfare Division of the U.S. Naval Research Laboratory) show a photograph of the USS Crockett and an image of the ship obtained from a U.S. Navy AN/APS-137 ISAR. [1]

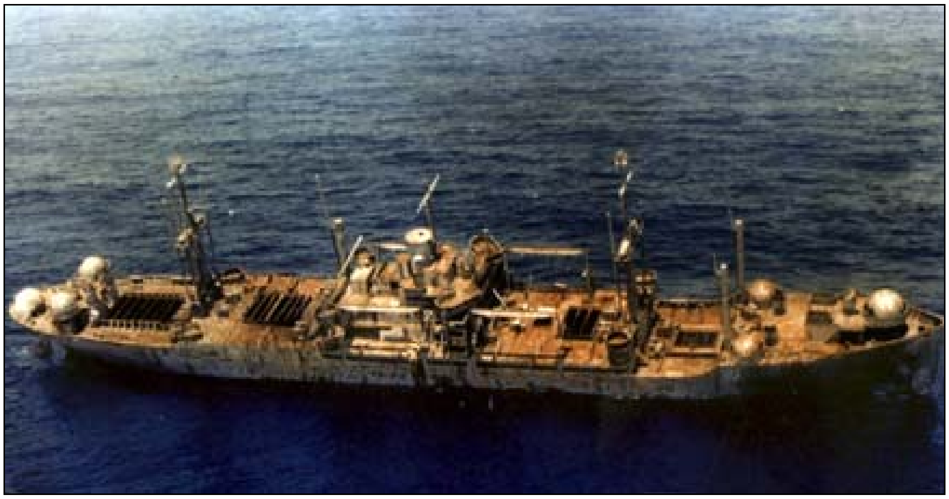


Figure 1. USS Crockett (From [1])

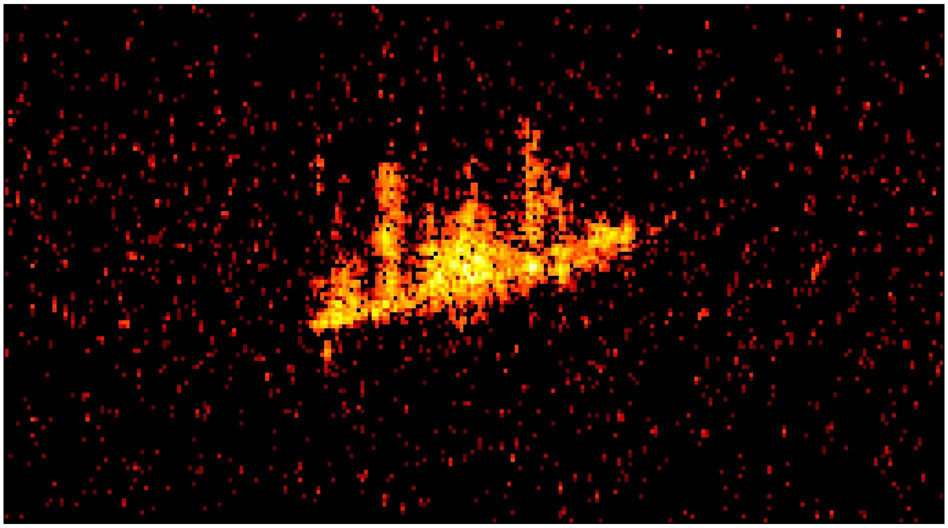


Figure 2. AN/APS-137 ISAR Image of the USS Crockett (From [1])

Explained in depth in the papers mentioned, ISAR can provide the target's range, bearing and positional data with both medium and high-resolution images for display and recording. It can also be used in launching weapon systems. For example, before a cruise missile is launched, the classification of the target may be pursued using an ISAR image. This image can be used for recognition and identification. Imaging capability is an advantage over previous technology because it improves the ability to identify the specific type of target, distinguish enemy from friend, guide the weaponry, and defeat the false target decoys. Depending on the target identification, the decision to engage the target and launch the missile is made, and only the ability to quickly confuse the ISAR targeting process will prevent the missile from being launched. [1, 2]

2. Countering ISAR and Digital Image Synthesizer (DIS)

Actions taken to confuse or deceive pre-launch weapons designation and targeting efforts are known as 'counter targeting techniques' and include use of low radar cross-section materials, stealth technology and pre-lock-on deception devices. Unfortunately, these techniques are ineffective against wideband imaging radars. [1]

As a result, modern wideband-imaging-ISARs create a difficult ship defense problem. [2] For example, if an adversary is using a wideband imaging ISAR, an electronic protection system cannot synthesize a false target by just transmitting a signal that emulates a radar return off a single or a few scattering surfaces. Instead, such a transmitted signal must emulate a coherent sequence of reflections with proper delay, phase, and amplitude that is similar to what would come from the multiple scattering surfaces at multiple ranges (distances from the radar) of an actual ship. Analog methods for generating false radar targets have included the use of acoustic charge transport (ACT) tapped delay lines and fiber optic tapped delay lines. ACT devices are no longer commercially available and also have limited bandwidth, making them impractical against wideband imaging radars. Optical devices are bulky and costly to manufacture, especially for the longer delay line lengths needed to synthesize a false target image of even a moderately sized ship. However, the equations and algorithms needed to digitally synthesize a false target radar image have evolved considerably over the last several years. With modern digital signal processing (DSP) techniques and advanced VLSI fabrication processes, it is now possible to digitally synthesize a realistic false target radar image of even a large warship, such as an aircraft carrier.

The digital image synthesizer reduces both the noise of the repeated signal and size of the system. [2] Compared to analog technology, it reduces the cost. The programmable design allows rapid and adaptive modifications of the system into different types of targets offering a low cost decoy capability while utilizing readily available modern digital radio frequency memories (DRFMs). Thanks to the recent advances in integrated circuit (IC) fabrication processes, such as sub micron complementary metal oxide semiconductor (CMOS) and bipolar CMOS (BiCMOS) technologies, it has become easy to achieve fast and dense custom ASICs. For these reasons, a programmable imaging architecture for countering ISARs by generating realistic false target signatures is realized with a custom digital ASIC integrated with DRFMs.

B. RELATED WORK

Many researchers have taken part in the design of the DIS chip. Initial design testing performed by Amundson [3] and Guillaume [4] is well documented in their theses. Kirin [5] designed the mask layout of the sine/cosine Lookup Table. Ozguvenc [6] created the original Range Bin Processor (RBP) design.

Le Dantec [7] evaluated the DIS performance under different parameters. Bergon [8] did the VHDL (VHSIC Hardware Description Language; VHSIC is an acronym for Very High-Speed Integrated Circuits) modeling and testing of up to 32 RBPs. Prof. Fouts provided the mask layout of the summation adder and the registers. Mattox [9] and Prof. Fouts redesigned the DIS high level architecture to use counter-clock flow pipelining. Altmeyer [10] designed the phase extraction circuit. This circuit is required to convert the 'I' and 'Q' values from the DRFM to a usable 5–bit phase value that can be processed by the RBPs. Mattox also added special clock distribution circuits to allow a daisy chain clock distribution and a self-test unit. He also created the overall mask layout in accordance with the latest technology improvements and minor modifications in the design.

For additional information regarding the background of the DIS and the theory of operation, see References [1] and [2]. For additional information regarding the original and final designs of the RBP, refer to [6] and [7], respectively.

C. PRINCIPAL CONTRIBUTIONS

Research conducted within this thesis is mainly focused on the modeling, simulation and verification of the custom ASIC DIS chip. The simulation and verification mentioned in this study and some design efforts made by other researchers were performed simultaneously. Simulations were performed with Aldec Active HDL™ versions 5.1 and 5.2. Components such as inverters, registers, pass-gates, adders, multiplexers, the single RBP, 4 RBPs, 8 RBPs, 16 RBPs, self-test and phase extraction circuitry were tested individually and a final simulation performed with all components connected together.

Figure 3 shows the task flow followed to test and to verify the DIS chip The VHDL files were extracted directly from the schematic via Tanner Tools Pro S-Edit and supplied by the design team. Some modifications were made in order to comply with the naming conventions of Active HDL™ and behavioral descriptions for some components were added.

The error-free VHDL code was simulated using waveform tools in Active HDL™. The data flow was traced through the pipelined structure by monitoring the values on the schematic extracted from the VHDL code. Net names in the original circuit were identified as necessary to trace any discrepancy between expected and obtained results by using Tanner Pro S-Edit.

The outputs were compared to the results obtained from Tanner T-Spice simulation results and C++ calculations of the output values, both of which were supplied by Prof. Fouts. Testing the cascade of 128 RBPs and 512 RBPs could not be conducted due to software limitations in the memory allocation process during elaboration of the simulation. Chapter V contains more information on this issue.

.

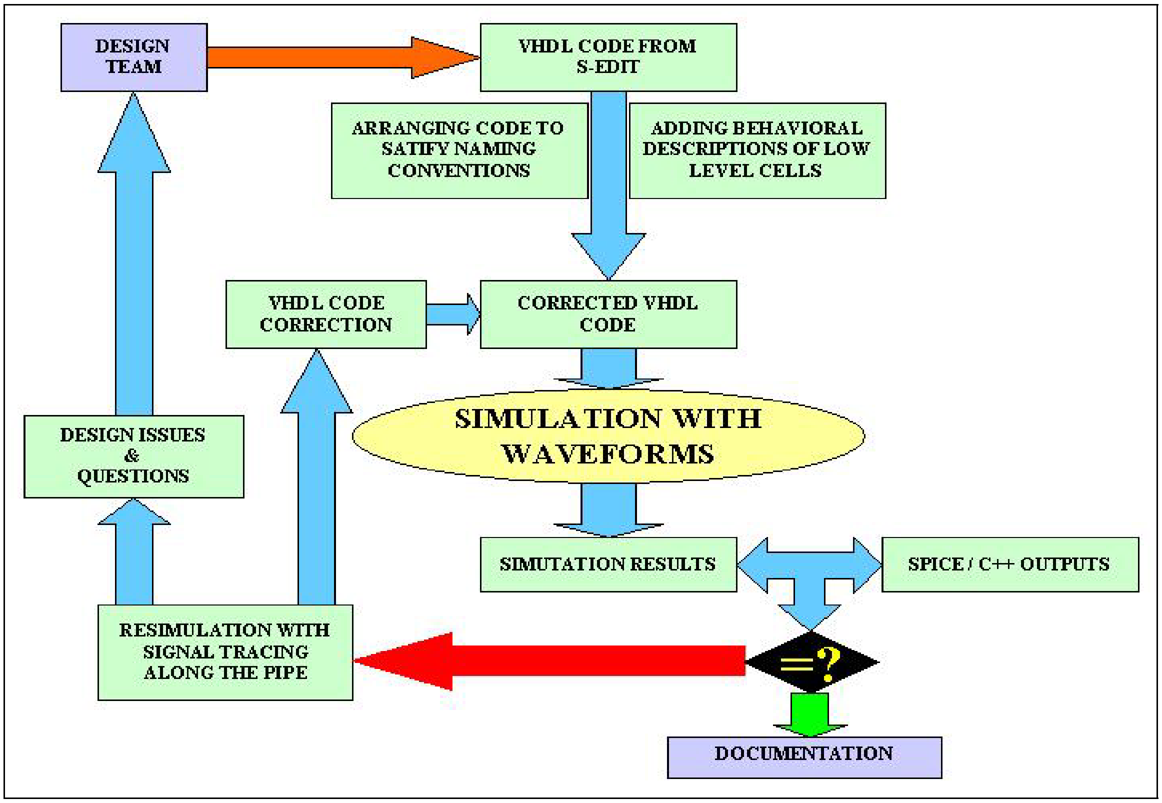

DESIGN TEAM
VHDL CODE FROM S-EDIT
ARRANGING CODE TO SATIFY NAMING CONVENTIONS
ADDING BEHAVIORAL DESCRIPTIONS OF LOW LEVEL CELLS
VHDL CODE CORRECTION
CORRECTED VHDL CODE
SIMULATION WITH WAVEFORMS
DESIGN ISSUES & QUESTIONS
SIMUTATION RESULTS
SPICE / C++ OUTPUTS
RESIMULATION WITH SIGNAL TRACING ALONG THE PIPE
=?
DOCUMENTATION

Figure 3. Testing and Verification Flow

D. THESIS OUTLINE

This thesis documents the testing and verification of the full-custom ASIC chip, including cascaded 512 range bin processors, phase extraction, self-test logic and other required circuitry.

Chapter II outlines more detailed information about the DIS chip and main components.

Chapter III presents the capabilities of VHDL as a means to design and/or verify a digital circuit design and contains some information about the software used, Aldec Active HDL^{TM}.

Chapter IV presents the simulations and the results of the low level cells used to construct the DIS chip.

Chapter V shows the simulations performed at the main functional blocks and overall DIS chip.

Chapter VI summarizes the results of the thesis, key lessons learned and recommendations for future work.

Appendix A contains the sequence of phase samples, which are generated by the Self Test Circuit to test the functionality of the DIS.

II. DIS CHIP

This chapter discusses the theory behind the idea of countering ISARs and explains the hardware implementation of the Digital Image Synthesizer (DIS). It also outlines the main functional components, such as the 512 RBP block, the Self Test Logic, the Phase Extractor and the Control Circuitry. The information on the fabrication technology is also presented.

A. THEORY

As shown in Figure 4, the DIS chip generates false target images from a series of intercepted Inverse Synthetic Aperture Radar (ISAR) chirp pulses to provide an imaging decoy capability. A Digital Radio Frequency Memory (DRFM) samples the phase and stores the intercepted ISAR pulses. An image synthesizer modulates the phase samples by synthesizing the temporal lengthening and the amplitude modulation caused by the many recessed and reflective surfaces of a target and generates a realistic Doppler profile for each surface. This digital signature is then converted into an analog signal and transmitted to the ISAR after being up-converted. [2]

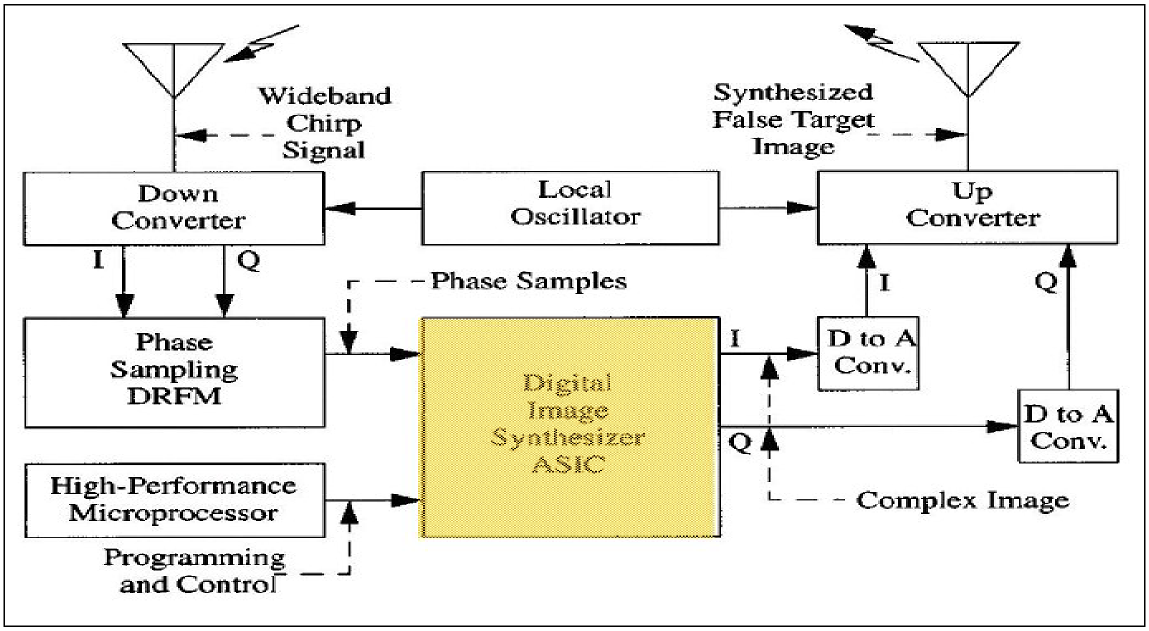


Figure 4. Overall System Block Diagram (From [2])

The DIS Application Specific Integrated Circuit (ASIC) contains a parallel array of identical complex digital modulators with one modulator for each false target range bin. Each binary phase sample is applied one at a time to the modulator array. Each range bin has a set of gain and phase coefficients that are derived from the range-Doppler description of the false target to be synthesized and a phase adder, a look-up table (LUT) and a summation adder. [2]

The single RBP data flow and implementation method are visualized in Figure 5. Each DRFM phase sample within a radar pulse is added to the phase coefficient to increment the phase and, therefore, accomplishes a phase rotation. This function is implemented with a binary adder, resulting in the desired motion profile of the range bin. In order to change the range bin's radar cross-section (RCS) characteristics, a rotated phase value is converted to a normalized complex signal (In-phase (I) and quadrature (Q)), using a lookup table (LUT). A gain circuit that multiplies the complex signal by a gain coefficient modulates the Radar Cross Section (RCS). Multiplication is implemented by left-shifting I and Q binary numbers using a parallel array of multiplexers. The last stage in the Range Bin Processor (RBP) is the summation of the gain block results with the adjacent (delayed) adder output and sending the results forward to the next RBP.

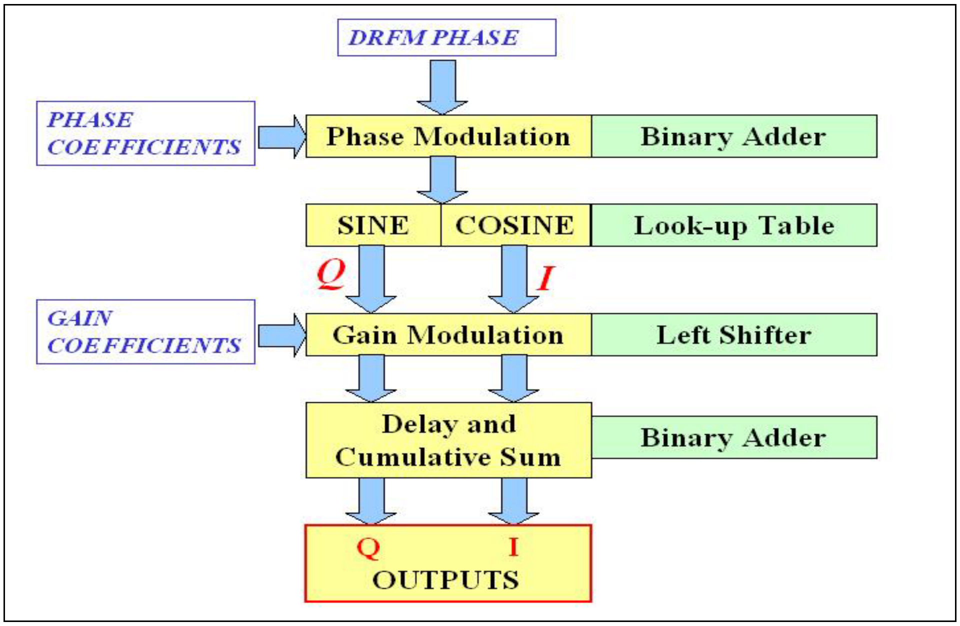


Figure 5. Single RBP Data Flow and Implementation Method

The output of the range bin processors is

$$I(m,n)=\sum_{r=0}^{N_r-1} 2^{g(r,n)}\, e^{i(\phi(n-r,n)+\phi_{inc}(r,n))} \quad , \tag{2.1}$$

where $2^{g(r,n)}$ is the gain multiplication coefficient factor and $e^{i(\phi(n-r,n)+\phi_{inc}(r,n))}$ is the phase of the signal, which includes $\phi_{inc}(r,n)$, the phase increment, added by each range bin processor. [1]

Each range bin processor computes a part of the final sum. The range bins are cascaded so that each adds its individual partial sum to the partial sum of previous processors. Double buffering of the programming data allows the processors to be programmed independent of the current sum they are computing. [9]

For additional information regarding the background of the DIS and the theory of operation, refer to [1] and [2].

B. HARDWARE IMPLEMENTATION

Overall, the DIS chip consists of 512 RBP s cascaded serially, self-test, phase extraction, and programming and control logic circuitry. Figure 6 shows the overall hardware block diagram. Four different set of phase samples can be steered into the RBP block using the control and programming inputs. The clock signal flows backwards with respect to the phase sample data flow direction. Cascaded RBPs produce the final I/Q output values using the phase and gain coefficients. Each RBP should be given these values separately prior to the introduction of the phase samples.

Figure 7 shows the actual schematic capture from S-Edit. In order to find detailed information on design parameters and S-Edit design process, refer to [9].

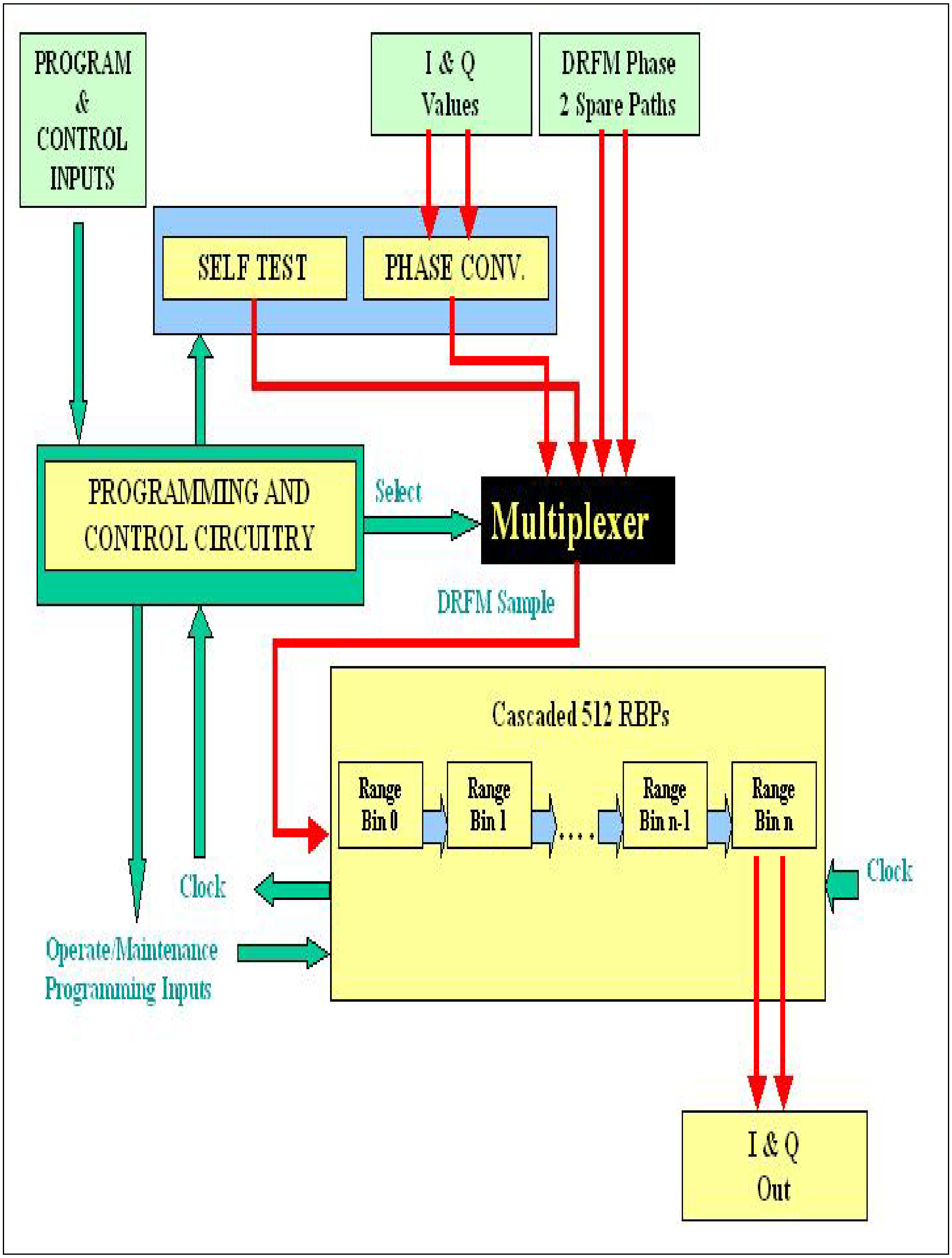


Figure 6. Whole DIS Virtual Hardware Implementation

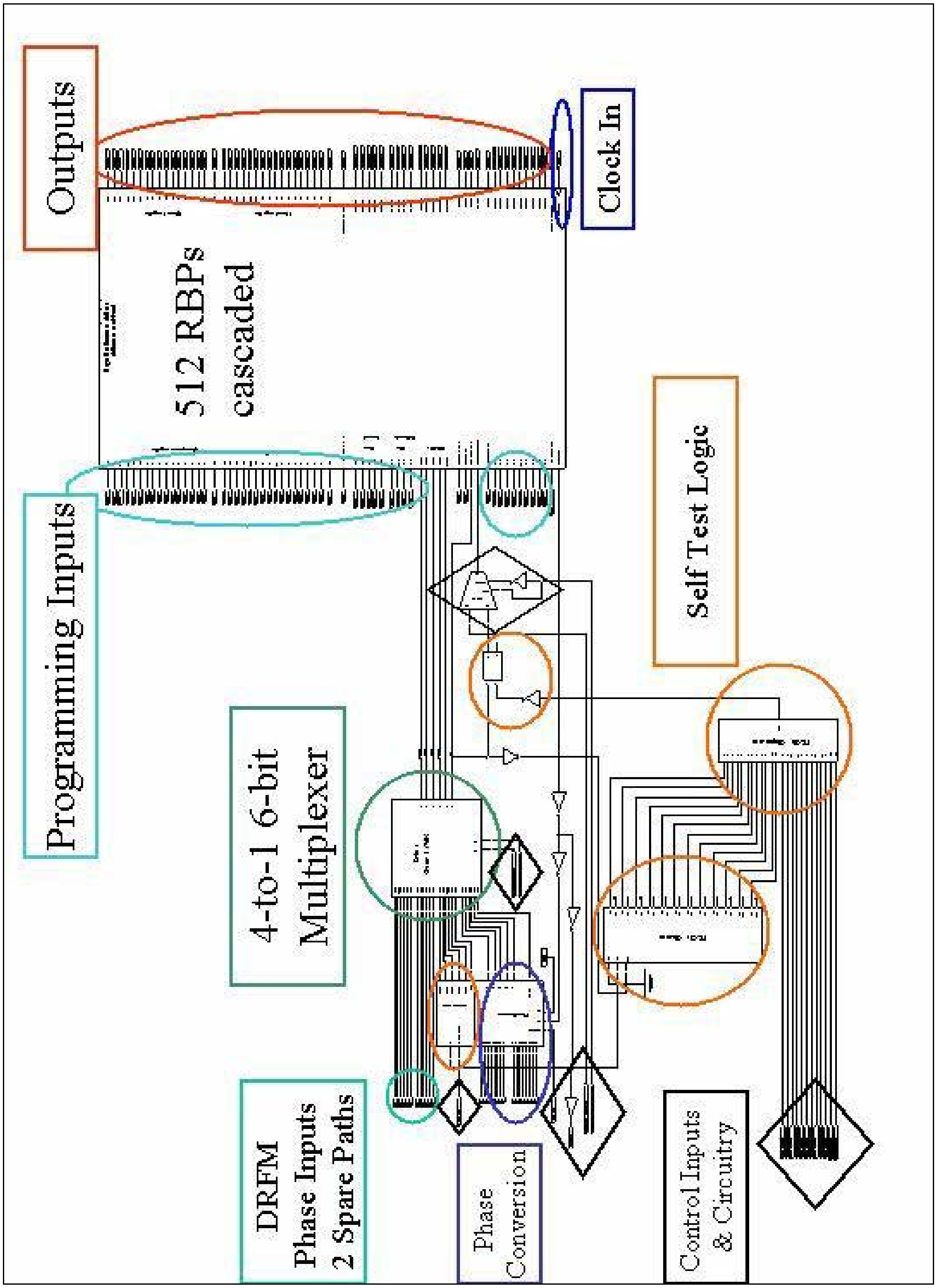


Figure 7. Complete DIS Hardware Implementation

1. 512-Range-Bin-Processor Block

This block is comprised of 512-range bin processors cascaded serially. The pipelining structure allows daisy chained clock distribution. The clock signal is propagated from the 512th RBP to the first RBP, which in turn conveys it to the phase extraction circuit and self-test circuit. As shown in Figure 8, each RBP calculates I/Q outputs and passes them to the next RBP to be added with the I/Q results from that RBP to generate the target profile.

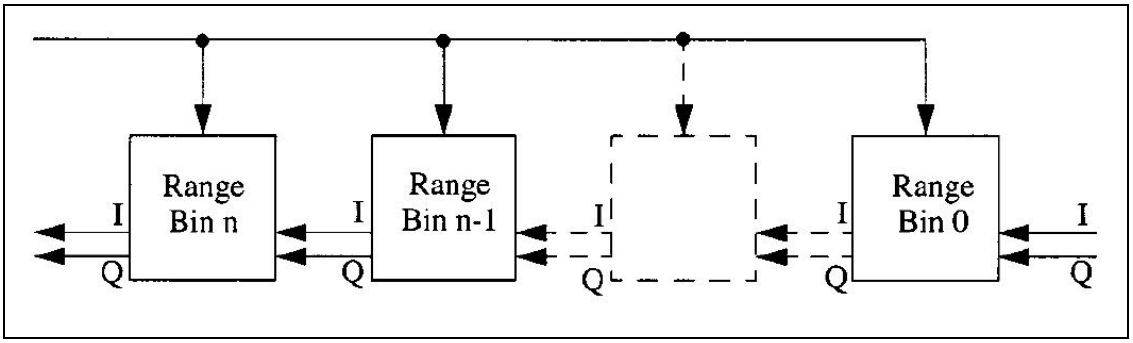


Figure 8. 512- Range-Bin-Processor Implementation (From [2])

Each Range Bin Processor is comprised of a phase adder, a look-up table (LUT), gain shifters, and final adders along with registers used for pipelining and pre-loading. The architecture of a single range bin processor can be seen in Figure 5, whereas the actual hardware implementation is presented in Figure 9.

Each RBP needs to be programmed with the phase increment and gain coefficients. This requires selectively programming them before the DRFM phase samples are fed to the RBP block. The address of each RBP, a 9-bit binary number, is hardwired into each RBP. For instance, address lines in RBP 0 are grounded whereas they are tied to VDD in RBP 511.

As select inputs and associated coefficients propagate in the pipe, they are compared with the address of each RBP. The matching RBP latches the proper coefficient values. A comparator and a preload register accomplish this function.

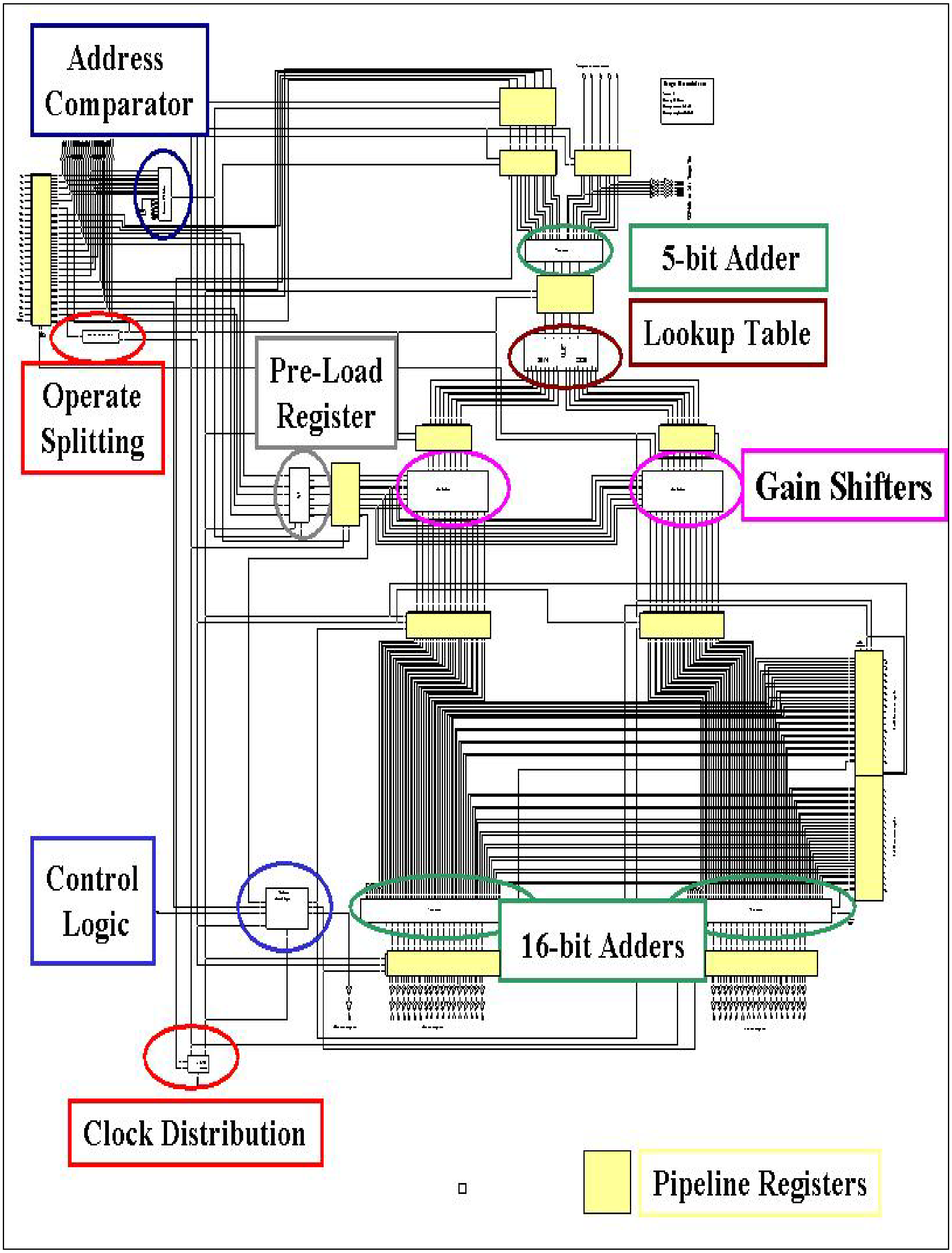


Figure 9. Single RBP Actual Hardware Implementation

The phase rotation adder generates the motion profile of the range bin by adding DRFM phase samples and phase increment values (PInc). The incremented phase values are converted to I /Q values by the Lookup Table (LUT).

I/Q values from the LUT are modulated with gain shifters by applying gain coefficients. The proper values are programmed by the control microprocessor. The gain shifters realize multiplication by powers of 2.

The I/Q values from previous range bins are then added to the computed I/Q values by using 16-bit adders. The sum is the final result if the range bin is the 511th RBP. Otherwise, the results are sent to the next consecutive RBP with the next clock.

2. Self Test Logic

The self-test logic is basically a linear feedback shift register (LFSR) counter, which can have $2^n - 1$ (in this case 4095) states. It is used to generate a maximum-length sequence of inputs. The pseudo random counting sequence of the LFSR is more likely to detect errors than a binary counting sequence. More information on the LFSR can be found in [11].

The self-test logic circuit implementation is shown in Figure 10. With the initialization of the sequence, one register is set and the others are cleared, which eliminates the all zero-valued-registers case. Therefore, as the self-test sequence is started, it generates Phase Sample Valid (PSV) and DRFM0 – DRFM4 outputs in a pseudo-random pattern. The outputs of the circuit can be monitored and compared with predicted results to detect any malfunction in the overall circuit. The outputs of the self-test logic and their use with the control circuitry will be discussed in Chapter V.

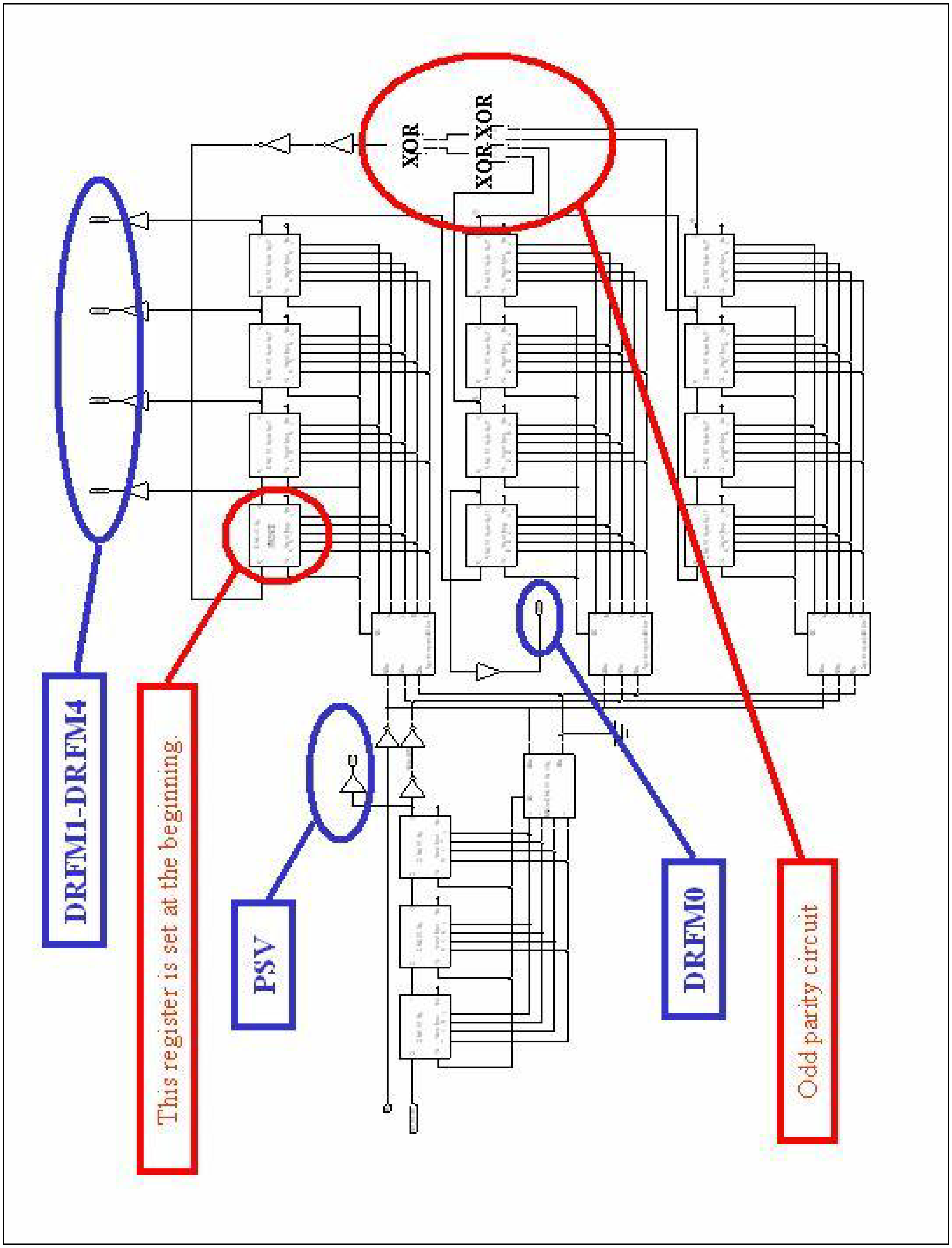


Figure 10. LFSR as a Self Test Sequence Generator

3. Phase Extractor

This circuit converts the I/Q values supplied by the DRFM as eight-bit two's complement numbers into corresponding phase angle values expressed as five-bit unsigned numbers for generating the false target signature. The detailed information on the conversion methodology and implementation can be found in [10].

4. Programming and Control Circuitry

This portion consists of the programming coefficient inputs for the range bin processors and select inputs to address a specific range bin to be programmed, a 4-to-1 6-bit multiplexer that steers the data from four sources (self test, phase extractor, two separate paths) into the 512-RBP block, a counter to determine the length of the self test sequence, and an S/R latch with a 2-to-1 multiplexer to switch the operating mode from/to operate to/from maintenance modes. Extra inputs are used to select the operating mode, the data path to be used, and start self-test sequence or phase extraction.

Figure 11 shows the control circuitry implementation in detail. The programming inputs can be seen in Figure 7.

5. Fabrication Technology

As presented by Mattox in [12], the proof-of-concept chip was manufactured with an 0.5 μm process and 81632 transistors, including I/O pads. It had 126 input/output pins and two ground and two VDD pins. It operated with a 3.3 V voltage supply at 70 MHz, consuming 0.132 Watts. It occupied 5.5 mm by 6.1 mm of area.

The design and technology used in the DIS has been greatly modified relative to the proof-of-concept chip in order to comply with the full specifications for the DIS and to benefit from technology improvements. Prof. Fouts and Mattox have completed the final design. Table 1 shows the information about the final design.

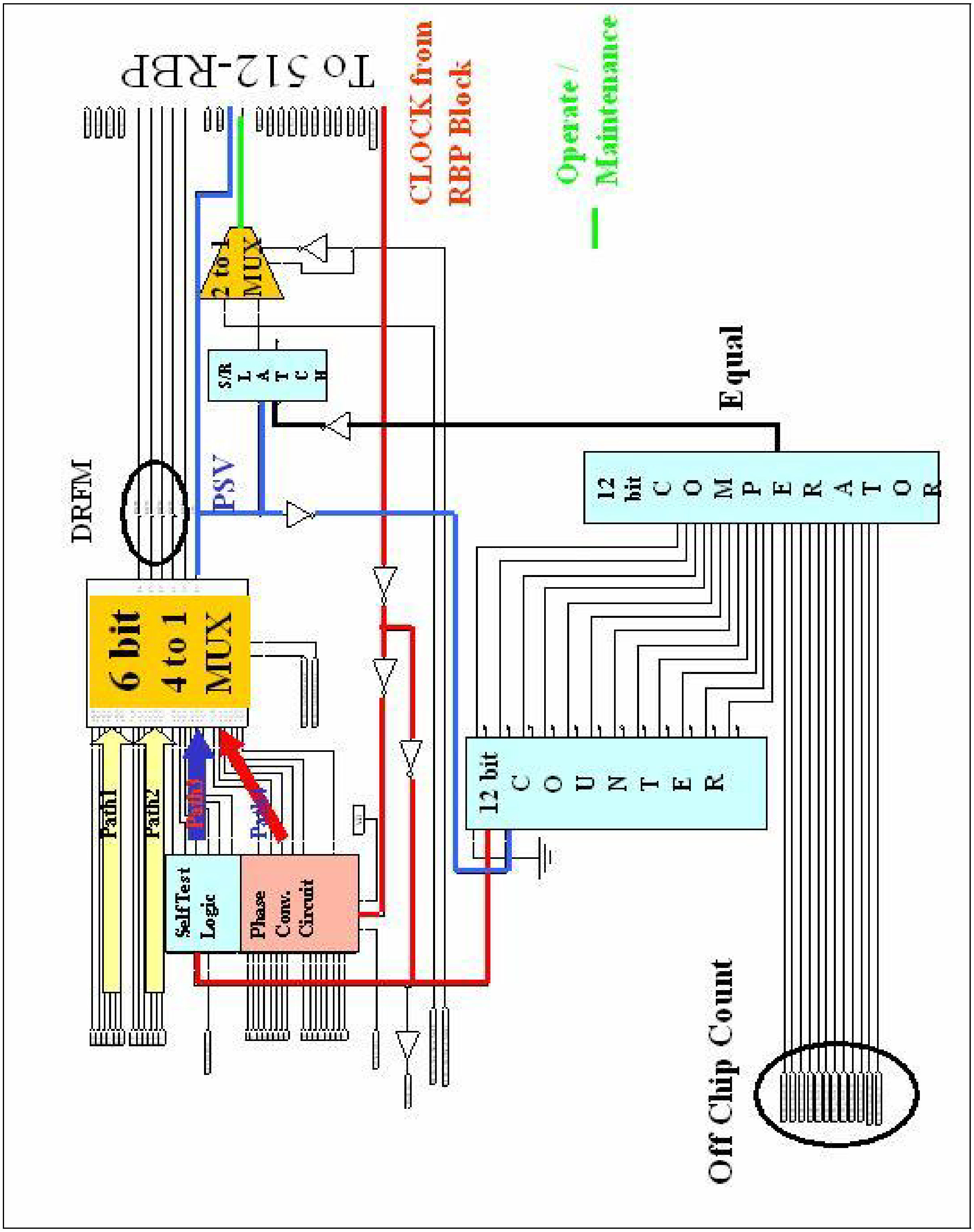


Figure 11. Control Circuitry Implementation

Process	**0.18 mm CMOS 6 metal process (TSMC)**
Physical dimension	**“9.0 mm x 7.9 mm” (minimum, RBP block only)**
Number of Transistors	**Over 5.5 Million Transistors**
Number of Pins	**130 I/O pins, Dozens of VDD/GND pins**
Power Consumption	**16.1 W at 700 MHz using 1.8-V supply**

Table 1. Specifications of the Final Chip (After [12])

The theory lying behind countering ISARs and the hardware implementation of the DIS discussed in this chapter forms the main subject of the design process. The testing and verification phases of the design process consist of using a hardware description language, VHDL, for simple and precise functional simulations of the DIS. Chapter III gives information on VHDL and the simulation software used to test the components.

III. INTRODUCTION TO VHDL HARDWARE DESCRIPTIVE LANGUAGE AND ACTIVE HDL™

This chapter contains basic information on Hardware Description Languages and the VHDL. It also introduces the software tool, Aldec Active HDLTM, used in VHDL modeling, functional simulation and verification of the DIS.

A. VHDL HARDWARE DESCRIPTIVE LANGUAGE

1. Background

The need for a standardized representation of digital systems to share designs of subsystems across contractors became apparent. To address this issue, the first version of VHDL was released in 1985 by a committee of the U.S. Department of Defense (DoD). The Institute of Electrical and Electronic Engineers (IEEE) standardized the language and released IEEE standard 1076-1987 in 1987. The latest version of the VHDL standard is IEEE 1076-1993. Drafts for a revised standard are currently in progress. [8]

2. Digital Design Using Hardware Descriptive Languages and VHDL

The digital systems design process starts from the specification of requirements and proceeds to produce a functional design. This design is then physically implemented through a sequence of steps. Like the full-custom Digital Image Synthesizer addressed in this thesis, a custom ASIC is generally the highest performing solution for any computation but often the most expensive and time consuming one. An example of the sequence of activities that typically take place during classical ASIC design is shown in Figure 12.

System requirements often consist of the function(s) to be realized, speed, power consumption, size and cost constraints. These functional requirements are then refined to a more detailed design description at the level of registers, memories, arithmetic units and state machines, which becomes the Register Transfer Level (RTL) of the design. Implementation of each RTL component produces the Logic Design of the system. Both RTL and logic level designs can be used to ensure that the design meets the original specifications. Fault simulations can be conducted to measure the effects of possible manufacturing defects on the chip and the environmental factors in which the chip is to be operated.

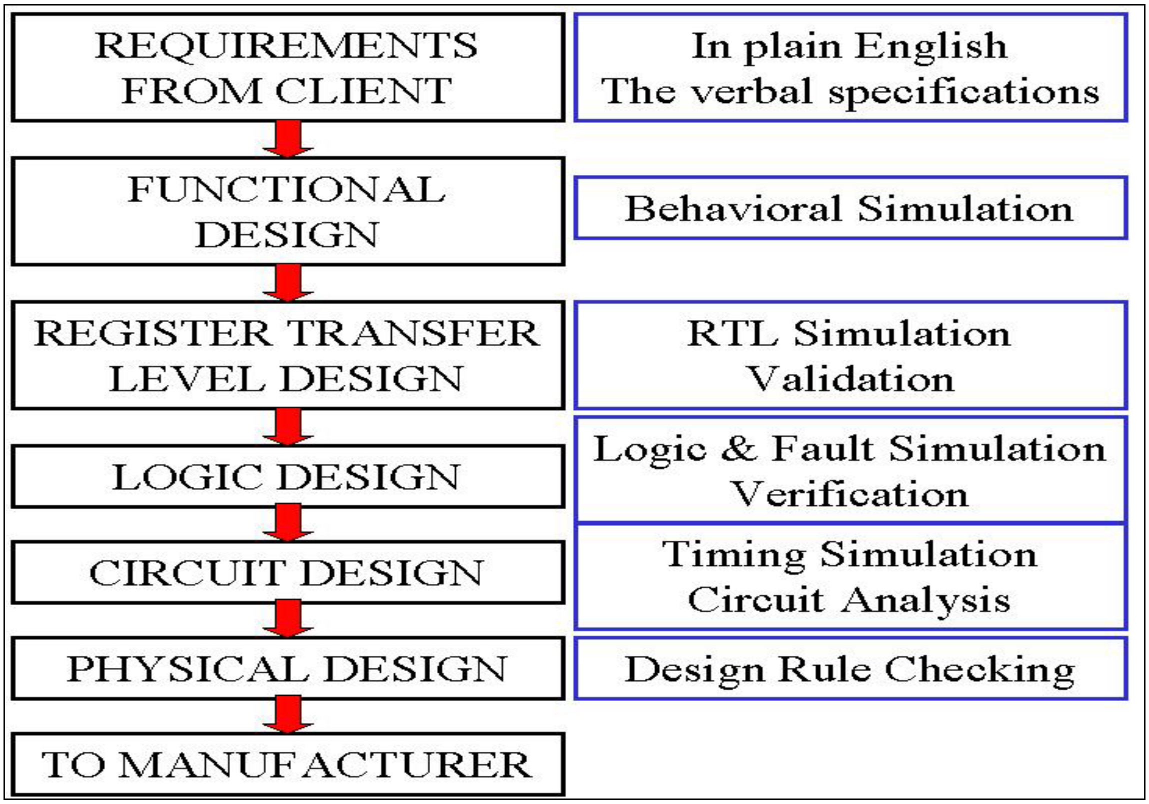

REQUIREMENTS FROM CLIENT
In plain English The verbal specifications
FUNCTIONAL DESIGN
Behavioral Simulation
REGISTER TRANSFER LEVEL DESIGN
RTL Simulation Validation
LOGIC DESIGN
Logic & Fault Simulation Verification
CIRCUIT DESIGN
Timing Simulation Circuit Analysis
PHYSICAL DESIGN
Design Rule Checking
TO MANUFACTURER

Figure 12. Activity Flow in Classic Digital System Design (After [13])

Finally, the logic level implementation is transformed into a circuit level implementation and physical chip layout. Design rule checks and circuit parameter extraction can be done at the physical design level.

At each level of this design hierarchy there are components that are used to describe the design. At the higher, or more abstract, levels there is a smaller number of more powerful components such as arithmetic units and memories. At the lower, and less abstract, levels there is a larger number of simpler, less-powerful components such as logic gates and transistors. Each level of the design hierarchy corresponds to a level of abstraction. The accuracy of simulation results increases at lower levels of hierarchy with the cost of longer simulation times.

In this classical approach, the design errors at low levels of detail are expensive to correct. They can also lead to a longer development time, which naturally increases the cost. The major drawback of traditional design methods is the manual translation of the design description into a set of logical equations. This step can be entirely eliminated with Hardware Description Languages (HDLs). With the ability of simulating circuits at different levels of abstraction, errors can be discovered and corrected early. [13]

In VHDL, designs can be decomposed hierarchically. Each design has not only a well-defined interface to connect it to other components but also a precise behavioral specification to simulate it. VHDL can be used to define behavioral specifications either in an algorithm or in actual hardware structure. For example, an algorithm can be used to stimulate a component to test higher levels of operation and it can be replaced with real hardware implementation later if the simulations are successful at higher levels. VHDL also allows concurrency, timing and clock modeling. It can also handle asynchronous circuits as well as synchronous sequential-circuit structures. Logical operation and timing behavior of a design can also be simulated.

In this thesis, the VHDL code of a full-custom Digital Image Synthesizer ASIC was generated automatically by a schematic capture editor, Tanner Pro S-Edit. Although it has the pictorial schematics of the components to provide the hardware design engineers a "sense and feel" of the design process, it lacks a logic level simulator. For simulation purposes, the code generated by Tanner Pro S-Edit was used in the Aldec Active HDL^{TM} tool. In this code, the components are defined in the structural domain, describing them in actual circuitry with minimum levels of abstraction to predict the system behavior as accurately as possible. Although the code generated is not optimum in size, it allowed a thorough testing and verification of each component and the overall circuit in Active HDL^{TM}.

B. VHDL CAPABILITIES OF ACTIVE HDL™

1. VHDL as a Programming Language in Active HDL™

The primary hardware abstraction in VHDL is the *entity*. It represents a part of the design with well-defined inputs and outputs and performs a well-defined function. *Entity* is the description of the interface between a design and its external environment. It may also specify the declarations and statements that are part of the design *entity*. A given *entity* declaration may be shared by many design *entities*, each of which has a different *architecture*. Thus, an *entity* declaration can potentially represent a class of design *entities*, each having the same interface. *Entity* declarations resemble software class descriptions.

Architecture body describes input output transformations and the internal composition or the behavior of the *entity* more like a software object. It is associated with an *entity* declaration to describe the internal organization or operation of a design entity. It is also used to describe the behavior, data flow, or structure of a design *entity*.

Signals provide the interactions between concurrent statements. *Signal* is an object with a past history of values. A *signal* may have multiple drivers, each with a current value and projected future values. The term *signal* refers to objects declared by *signal* declarations and *port* declarations.

A *component* describes a substructure of a design entity that is interconnected through signals. It represents an *entity/architecture* pair and specifies a subsystem, which can be *instantiated* in another *architecture* leading to a hierarchical specification.

A *process* statement defines an independent sequential *process* representing the behavior of some portion of the design. It consists of the sequential statements whose execution is made in the order defined by the user. During execution all concurrent statements are executed during the same simulation cycle and values of all modeled signals are computed. No VHDL model should depend on the order of execution of its concurrent statements. *Process* statements such as *case* and *loop* allow user defined sequential statements, which are beneficial especially in sequential circuits that have feedback loops for initialization purposes. When a *signal* takes on a new value, the sensitivity list of the concurrent statement decides if the statement is sensitive to that particular signal and acts accordingly.

As an example, the logic symbol and schematic representation of an inverter is shown in Figure 13. The code given in Figure 14 is generated by S-Edit and modified to supply the behavioral descriptions of n-type and p-type transistors.

As seen in the VHDL code, the descriptions of transistors and *entities* Vdd and Gnd (power supplies) are defined as *behavioral* descriptions in *architecture* body and they are *instanced* in the *entity* inverter.

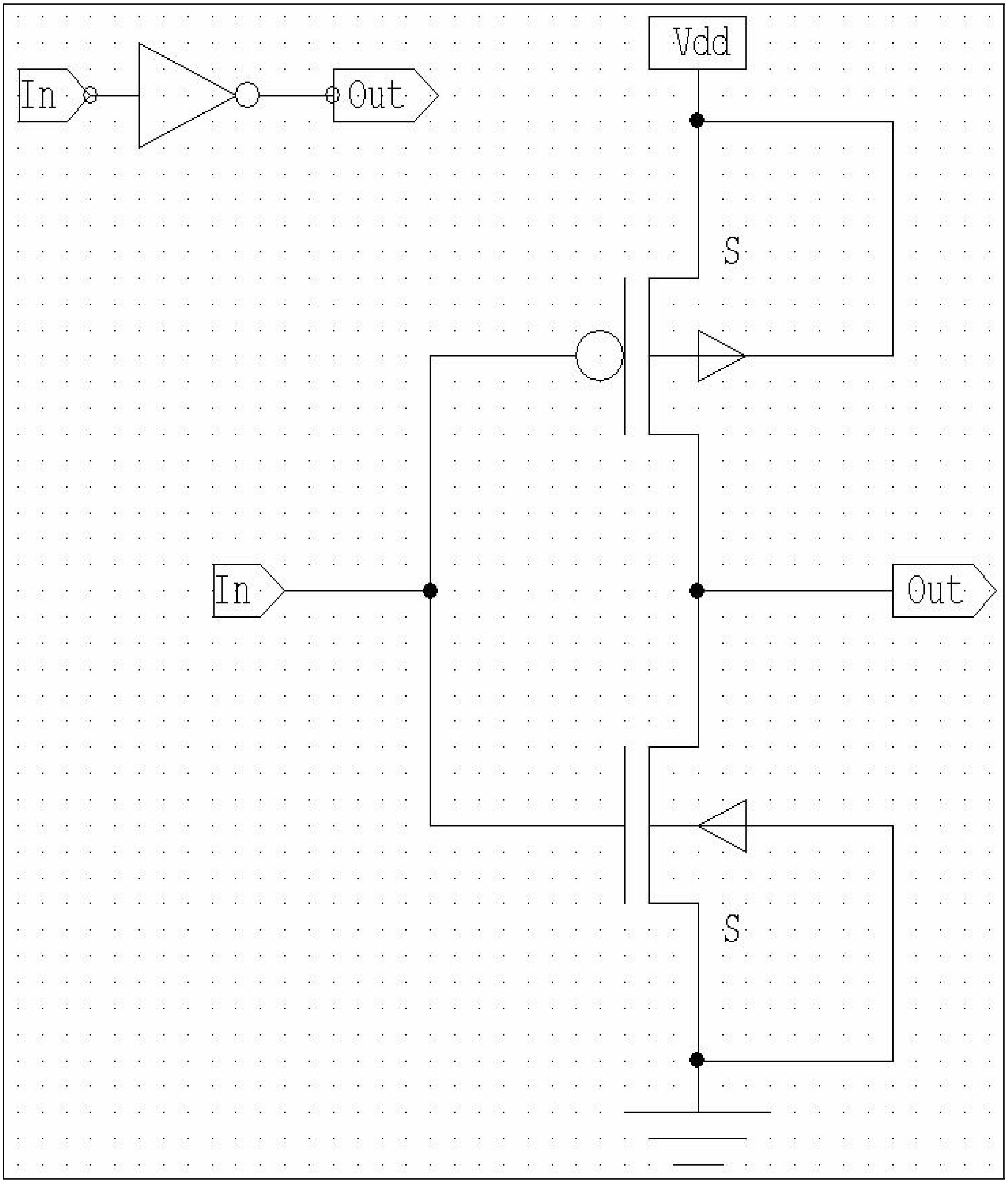
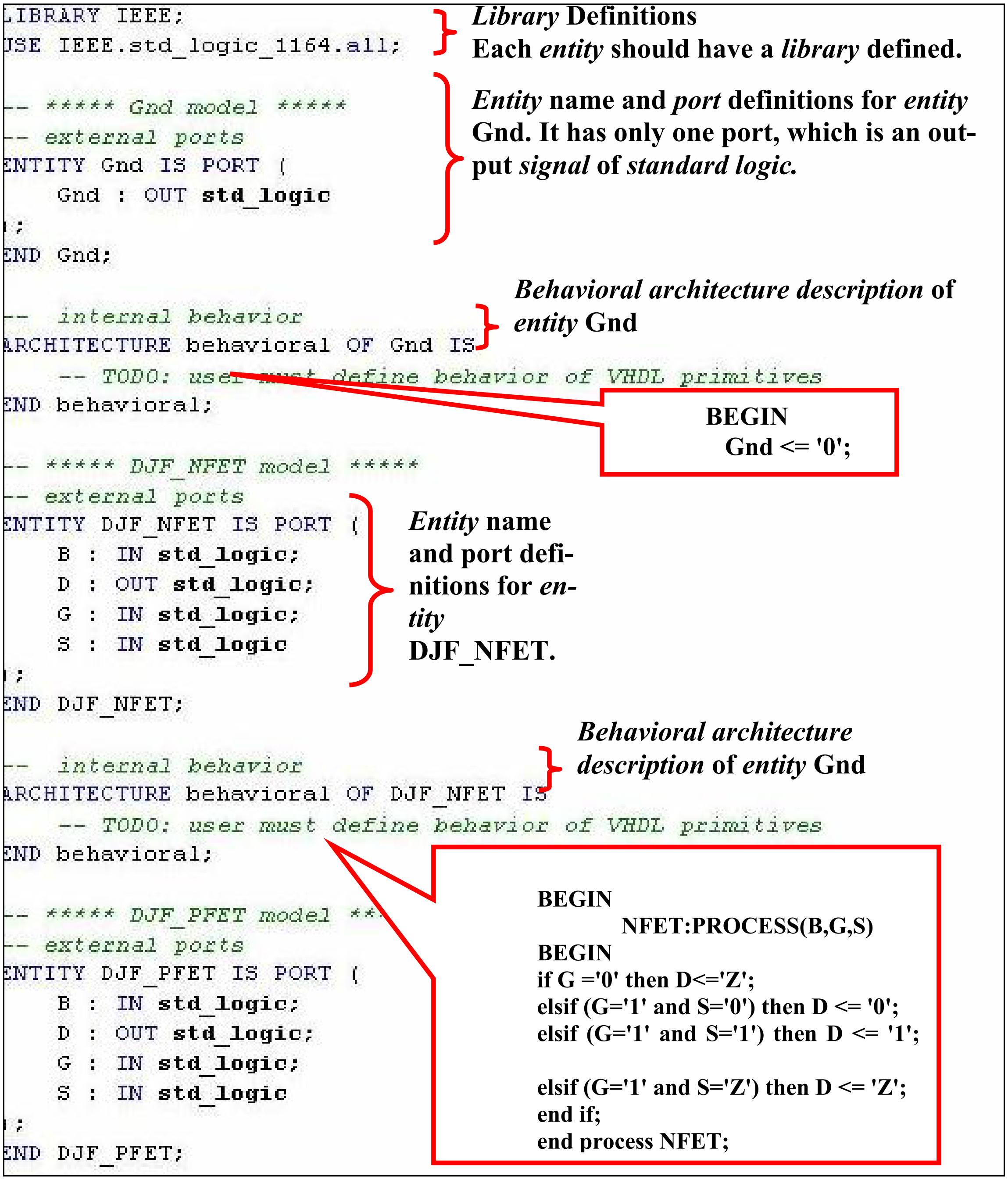


Figure 13. Logic Symbol and Schematic Representation of an Inverter in S-Edit



```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

-- ***** Gnd model *****
-- external ports
ENTITY Gnd IS PORT (
    Gnd : OUT std_logic
);
END Gnd;

--  internal behavior
ARCHITECTURE behavioral OF Gnd IS
    -- TODO: user must define behavior of VHDL primitives
END behavioral;

-- ***** DJF_NFET model *****
-- external ports
ENTITY DJF_NFET IS PORT (
    B : IN std_logic;
    D : OUT std_logic;
    G : IN std_logic;
    S : IN std_logic
);
END DJF_NFET;

--  internal behavior
ARCHITECTURE behavioral OF DJF_NFET IS
    -- TODO: user must define behavior of VHDL primitives
END behavioral;

-- ***** DJF_PFET model *****
-- external ports
ENTITY DJF_PFET IS PORT (
    B : IN std_logic;
    D : OUT std_logic;
    G : IN std_logic;
    S : IN std_logic
);
END DJF_PFET;
```

***Library* Definitions**
Each *entity* should have a *library* defined.

Entity* name and *port* definitions for *entity* Gnd. It has only one port, which is an output *signal* of *standard logic.

***Behavioral architecture description* of *entity* Gnd**

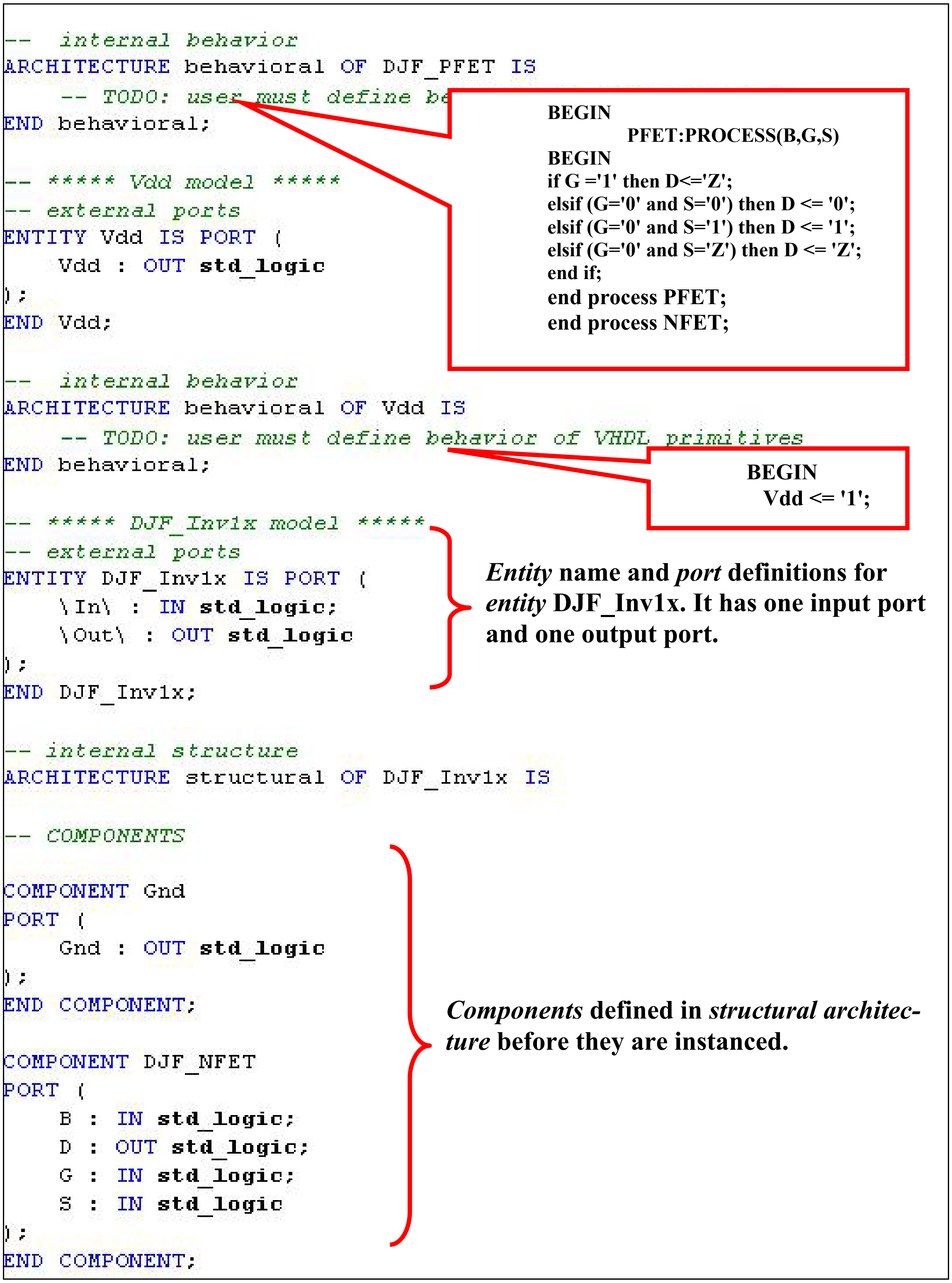
```
BEGIN
  Gnd <= '0';
```

***Entity* name and port definitions for *entity* DJF_NFET.**

***Behavioral architecture description* of *entity* Gnd**

```
BEGIN
        NFET:PROCESS(B,G,S)
BEGIN
if G ='0' then D<='Z';
elsif (G='1' and S='0') then D <= '0';
elsif (G='1' and S='1') then D <= '1';

elsif (G='1' and S='Z') then D <= 'Z';
end if;
end process NFET;
```

Figure 14.   Modified VHDL Code for an Inverter



```
--  internal behavior
ARCHITECTURE behavioral OF DJF_PFET IS
    -- TODO: user must define be
END behavioral;

-- ***** Vdd model *****
-- external ports
ENTITY Vdd IS PORT (
    Vdd : OUT std_logic
);
END Vdd;

--  internal behavior
ARCHITECTURE behavioral OF Vdd IS
    -- TODO: user must define behavior of VHDL primitives
END behavioral;

-- ***** DJF_Inv1x model *****
-- external ports
ENTITY DJF_Inv1x IS PORT (
    \In\  : IN std_logic;
    \Out\  : OUT std_logic
);
END DJF_Inv1x;

-- internal structure
ARCHITECTURE structural OF DJF_Inv1x IS

-- COMPONENTS

COMPONENT Gnd
PORT (
    Gnd : OUT std_logic
);
END COMPONENT;

COMPONENT DJF_NFET
PORT (
    B : IN std_logic;
    D : OUT std_logic;
    G : IN std_logic;
    S : IN std_logic
);
END COMPONENT;
```

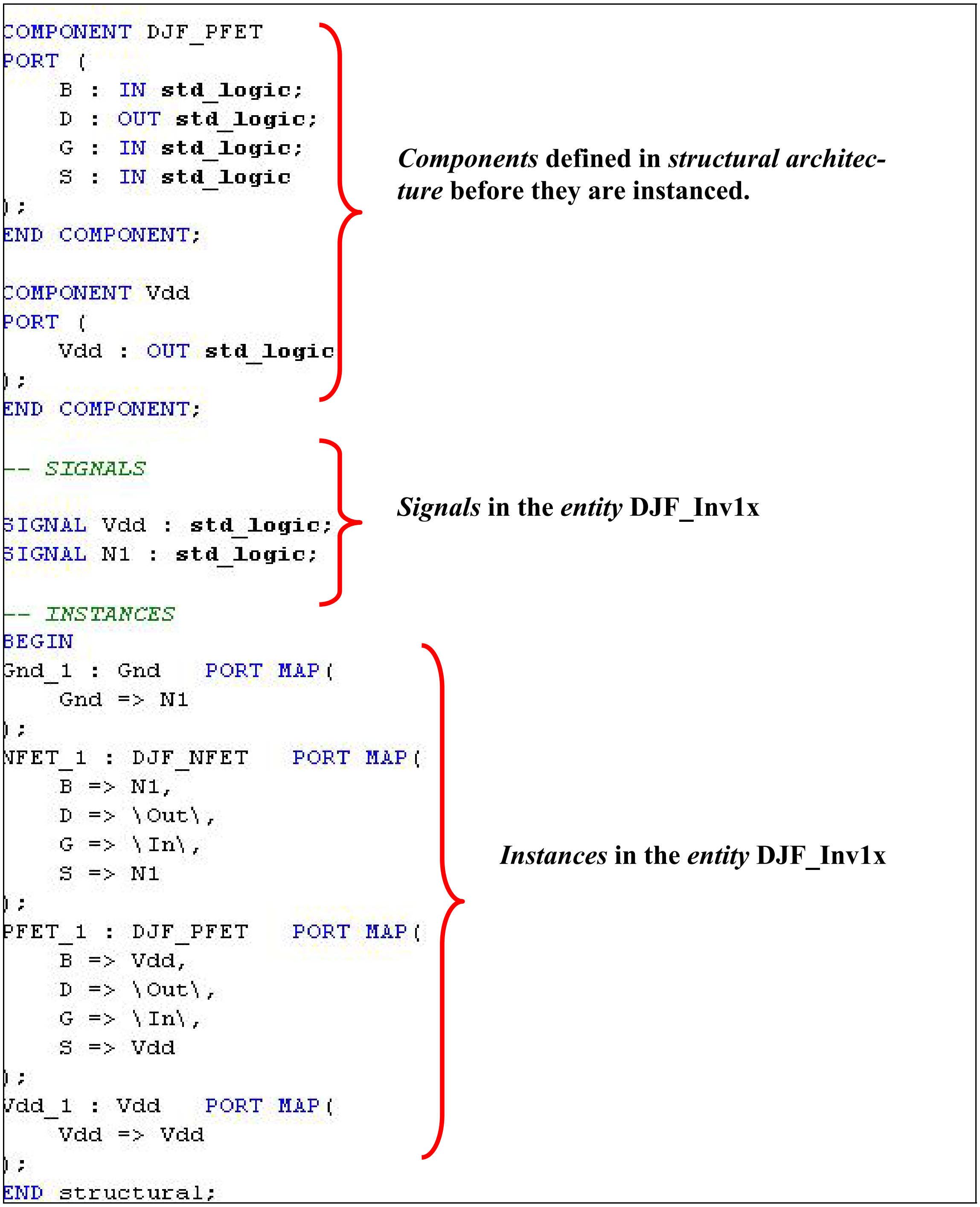
```
BEGIN
        PFET:PROCESS(B,G,S)
BEGIN
if G ='1' then D<='Z';
elsif (G='0' and S='0') then D <= '0';
elsif (G='0' and S='1') then D <= '1';
elsif (G='0' and S='Z') then D <= 'Z';
end if;
end process PFET;
end process NFET;
```

```
BEGIN
  Vdd <= '1';
```

Entity **name and** ***port*** **definitions for** ***entity*** **DJF_Inv1x. It has one input port and one output port.**

Components **defined in** ***structural architecture*** **before they are instanced.**

Modified VHDL Code for an Inverter, Continued



```
COMPONENT DJF_PFET
PORT (
    B : IN std_logic;
    D : OUT std_logic;
    G : IN std_logic;
    S : IN std_logic
);
END COMPONENT;

COMPONENT Vdd
PORT (
    Vdd : OUT std_logic
);
END COMPONENT;

-- SIGNALS

SIGNAL Vdd : std_logic;
SIGNAL N1 : std_logic;

-- INSTANCES
BEGIN
Gnd_1 : Gnd   PORT MAP(
    Gnd => N1
);
NFET_1 : DJF_NFET   PORT MAP(
    B => N1,
    D => \Out\,
    G => \In\,
    S => N1
);
PFET_1 : DJF_PFET   PORT MAP(
    B => Vdd,
    D => \Out\,
    G => \In\,
    S => Vdd
);
Vdd_1 : Vdd   PORT MAP(
    Vdd => Vdd
);
END structural;
```

***Components* defined in *structural architecture* before they are instanced.**

***Signals* in the *entity* DJF_Inv1x**

***Instances* in the *entity* DJF_Inv1x**

Modified VHDL Code for an Inverter, Continued

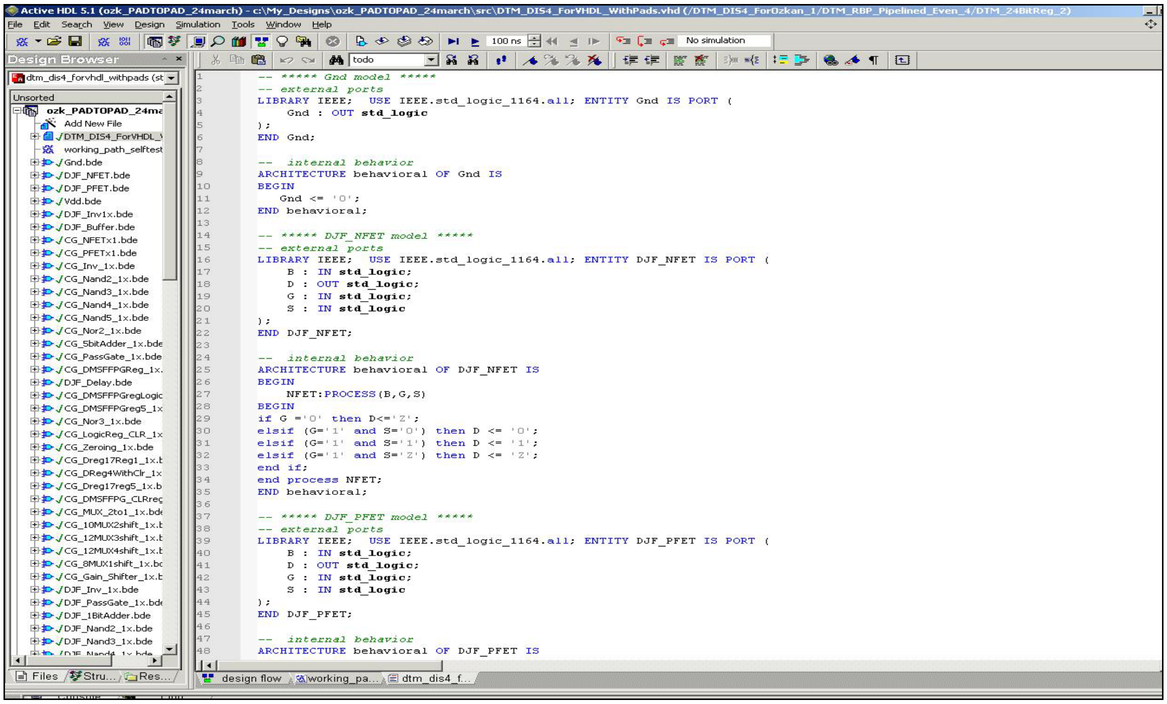
2. About Active HDL^{TM}

Aldec, Inc, of Henderson, NV, developed the tool chosen to perform the VHDL simulations, Active HDL^{TM} 5.1. It provides a number of useful features for development as well as testing hardware components. Its simulation technology supports IEEE VHDL 1076-1987/1993 and IEEE Verilog 1364-1995. Furthermore, it also supports EDIF 2.0.0 and single (VHDL or Verilog) or mixed (VHDL and Verilog together) language configurations.

This tool allows the user to create a design with three different methods. The first one, Text Editor, can be used to manually write the VHDL code or to copy a code into the design. The editor provides colorful representations of different syntax structures and makes programming easier. The second method, Block Diagram Editor, can be used to generate graphical symbols for gates and logic elements as well as to connect them for building larger structures. It provides visual assistance for the design engineer. The last method is the Finite State Machine Editor, which can be used to graphically creating designs using state diagrams.

The Active HDL^{TM} Text Editor resembles any text editor used for high level programming languages, such as C++. This environment is tightly integrated with the compiler and the simulator, which provides debugging capabilities. It also supplies the user with a built-in language assistance, automatic design structure generation capability, and setting or clearing of code breakpoints and cross probing of error messages. From the VHDL code, Active HDL^{TM} can generate block diagrams or finite state machines. Figure 15 shows the Text Editor.

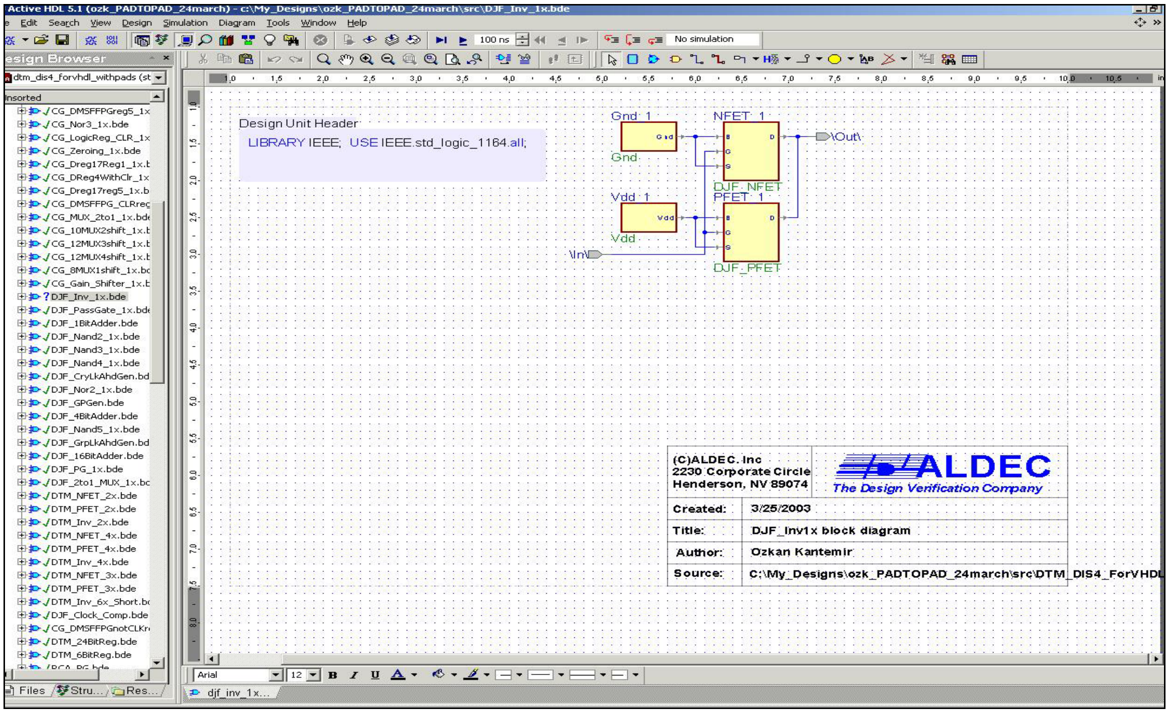
The Block Diagram Editor is a graphical representation of each entity in the VHDL code including signals and nodes. Active HDL^{TM} has a built-in library from different vendors to create schematics. The user can define and save new components and create a library. The Block Diagram Editor can export EDIF schematics as well as have a Design Rule Checking (DRC) capability. When compiled, it can generate the source code, which is executable. Figure 16 shows the Block Diagram Editor.



Active HDL 5.1 (ozk_PADTOPAD_24march) - c:\My_Designs\ozk_PADTOPAD_24march\src\DTM_DIS4_ForVHDL_WithPads.vhd (/DTM_DIS4_ForOzkan_1/DTM_RBP_Pipelined_Even_4/DTM_24BitReg_2)
File Edit Search View Design Simulation Tools Window Help
100 ns
No simulation
Design Browser
todo
design flow
working_pa...
dtm_dis4_f...
Files
Stru...
Res...

```
-- ***** Gnd model *****
-- external ports
LIBRARY IEEE;  USE IEEE.std_logic_1164.all; ENTITY Gnd IS PORT (
    Gnd : OUT std_logic
);
END Gnd;

-- internal behavior
ARCHITECTURE behavioral OF Gnd IS
BEGIN
   Gnd <= '0';
END behavioral;

-- ***** DJF_NFET model *****
-- external ports
LIBRARY IEEE;  USE IEEE.std_logic_1164.all; ENTITY DJF_NFET IS PORT (
    B : IN std_logic;
    D : OUT std_logic;
    G : IN std_logic;
    S : IN std_logic
);
END DJF_NFET;

-- internal behavior
ARCHITECTURE behavioral OF DJF_NFET IS
BEGIN
    NFET:PROCESS(B,G,S)
BEGIN
if G ='0' then D<='Z';
elsif (G='1' and S='0') then D <= '0';
elsif (G='1' and S='1') then D <= '1';
elsif (G='1' and S='Z') then D <= 'Z';
end if;
end process NFET;
END behavioral;

-- ***** DJF_PFET model *****
-- external ports
LIBRARY IEEE;  USE IEEE.std_logic_1164.all; ENTITY DJF_PFET IS PORT (
    B : IN std_logic;
    D : OUT std_logic;
    G : IN std_logic;
    S : IN std_logic
);
END DJF_PFET;

-- internal behavior
ARCHITECTURE behavioral OF DJF_PFET IS
```

Figure 15. Text Editor in Active HDL^{TM}



Active HDL 5.1 (ozk_PADTOPAD_24march) - c:\My_Designs\ozk_PADTOPAD_24march\src\DJF_Inv_1x.bde
Edit Search View Design Simulation Diagram Tools Window Help
100 ns
No simulation
esign Browser
Gnd_1
Gnd
NFET_1
DJF_NFET
Vdd_1
Vdd
PFET_1
DJF_PFET
\In\
\Out\
Arial
12
djf_inv_1x...

Design Unit Header
LIBRARY IEEE; USE IEEE.std_logic_1164.all;

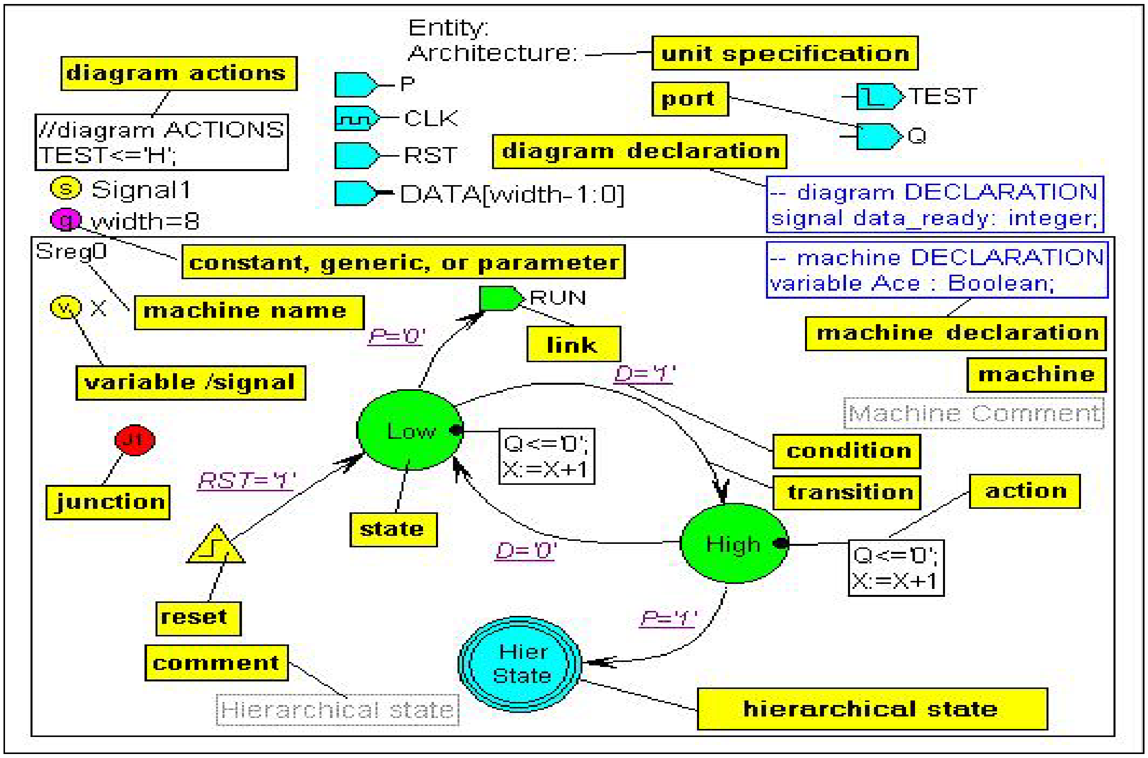
(C)ALDEC, Inc
2230 Corporate Circle
Henderson, NV 89074

ALDEC
The Design Verification Company

Created:	3/25/2003
Title:	DJF_Inv1x block diagram
Author:	Ozkan Kantemir
Source:	C:\My_Designs\ozk_PADTOPAD_24march\src\DTM_DIS4_ForVHDL

Figure 16. Block Diagram Editor in Active HDL^{TM}

The Finite State Machine Editor allows the user to enter a state diagram-based design. This diagram can be converted into VHDL or Verilog code for simulation and debugging purposes. HDL can be used with state diagrams. Figure 17 shows an example of a state diagram in Active HDL^{TM}. More information on creating a design, generating test benches, the waveform editor and utilizing the editors mentioned above can be found at [14].



```
//diagram ACTIONS
TEST<='H';
```

```
-- diagram DECLARATION
signal data_ready: integer;
```

```
-- machine DECLARATION
variable Ace : Boolean;
```

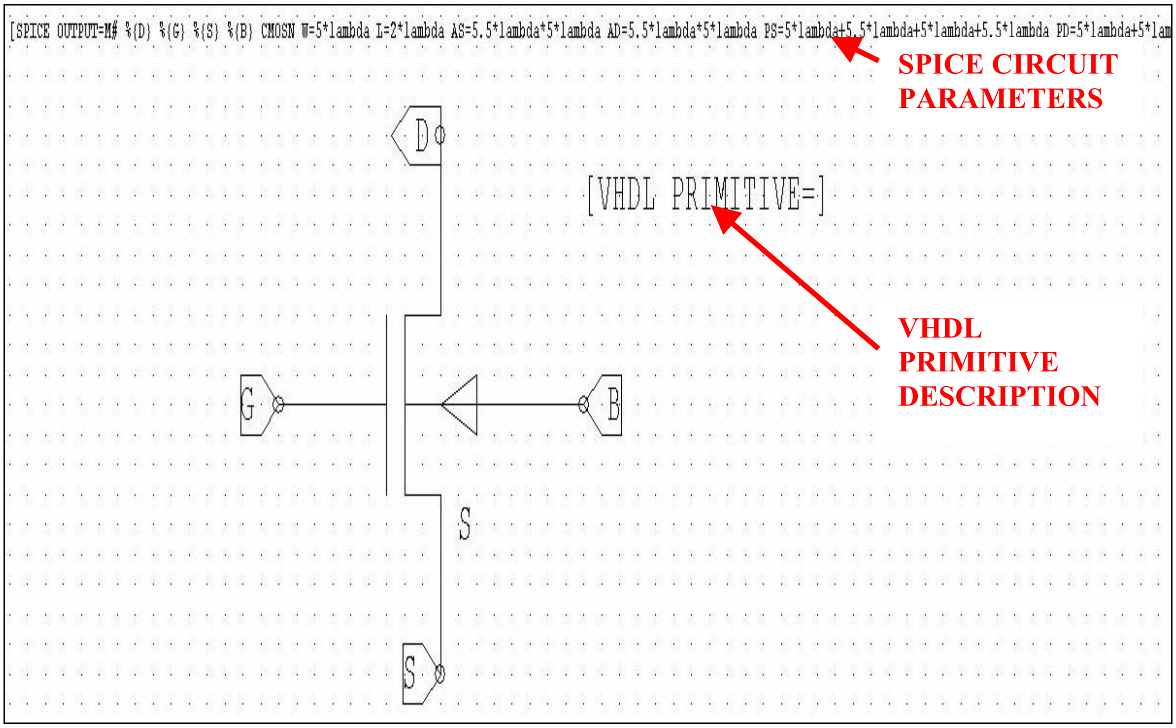
```
Q<='0';
X:=X+1
```

```
Q<='0';
X:=X+1
```

Figure 17. Finite State Machine Editor in Active HDL^{TM}

3. Test and Verification Methodology

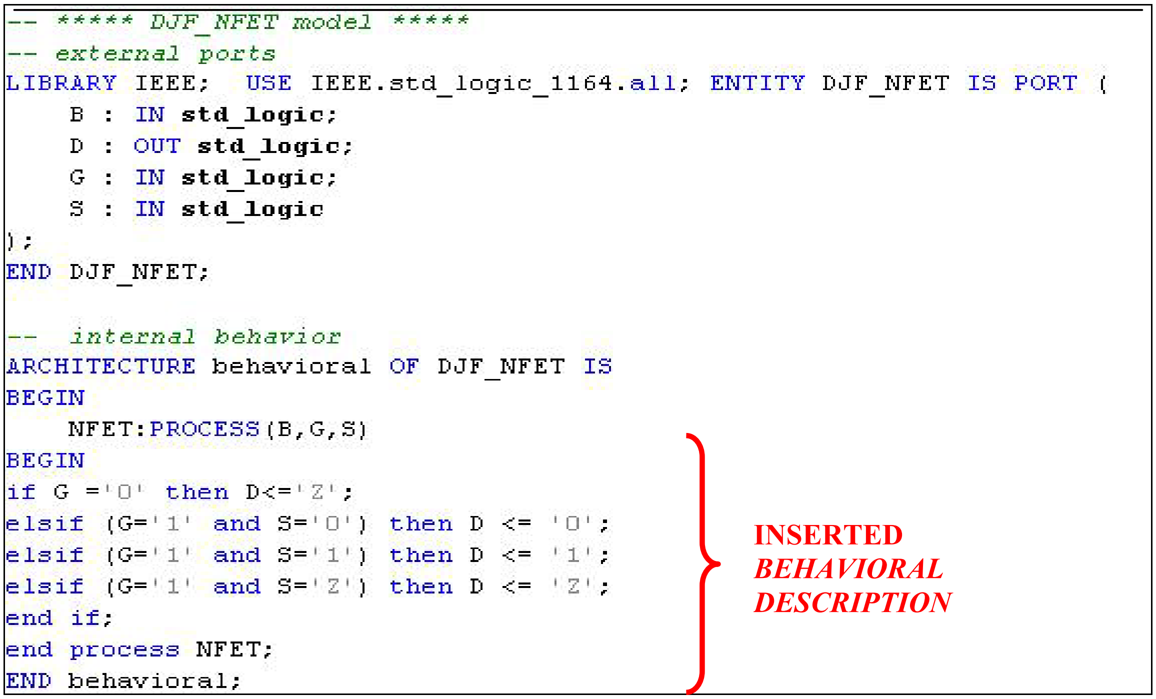
The VHDL code generated by the Tanner Pro S-EDIT tool contained some parts that need to be modified. First, the basic cells defined as VHDL primitives in S-EDIT should be re-defined in Active HDL^{TM}. For instance, the schematic representation of the *entity* DJF_NFET is shown in Figure 18 while the generated code for it and inserted behavioral description are in Figure 19. The *entity* is connected to other components via the ports shown in the schematic. The user should insert its *behavioral* description. Table 2 gives the *behavioral descriptions* used for the lower level *entities*.


[SPICE OUTPUT=M# %{D} %{G} %{S} %{B} CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda PD=5*lambda+5*lam
[VHDL PRIMITIVE=]
D
G
B
S
S

SPICE CIRCUIT PARAMETERS

VHDL PRIMITIVE DESCRIPTION

Figure 18. Schematic Representation of *Entity* DJF_NFET in S-EDIT



```
-- ***** DJF_NFET model *****
-- external ports
LIBRARY IEEE;  USE IEEE.std_logic_1164.all; ENTITY DJF_NFET IS PORT (
    B : IN std_logic;
    D : OUT std_logic;
    G : IN std_logic;
    S : IN std_logic
);
END DJF_NFET;

--  internal behavior
ARCHITECTURE behavioral OF DJF_NFET IS
BEGIN
    NFET:PROCESS(B,G,S)
BEGIN
if G ='0' then D<='Z';
elsif (G='1' and S='0') then D <= '0';
elsif (G='1' and S='1') then D <= '1';
elsif (G='1' and S='Z') then D <= 'Z';
end if;
end process NFET;
END behavioral;
```

INSERTED *BEHAVIORAL DESCRIPTION*

Figure 19. Generated Code and Inserted B*ehavior* for E*ntity* DJF_NFET

Entity		***Behavioral Description***

Ground	Gnd	begin
		Gnd <= '0';
		end behavioral;

Power Supply	Vdd	begin
		Vdd <= '1';
		end behavioral;

n-type transistors	NFET s	begin
		NFET:PROCESS(B,G,S)
		begin
		if G ='0' then D<='Z';
		elsif (G='1' and S='0') then D <= '0';
		elsif (G='1' and S='1') then D <= '1';
		elsif (G='1' and S='Z') then D <= 'Z';
		end if;
		end process NFET;
		end behavioral;

p-type transistors	PFET s	begin
		PFET:PROCESS(B,G,S)
		begin
		if G ='1' then D<='Z';
		elsif (G='0' and S='0') then D <= '0';
		elsif (G='0' and S='1') then D <= '1';
		elsif (G='0' and S='Z') then D <= 'Z';
		end if;
		end process PFET;
		end behavioral;

Delay Element	DJF_Delay_Element	begin
		Out_Delay <= In_Delay after 1 ps;
		end behavioral;

SR Latch	DTM_FfnotSnotR	Please Refer to Chapter IV Section B.

Table 2. Inserted VHDL *Behavioral Descriptions* for *Entities*

The use of Find/Replace and other utilities in the Text Editor eases the insertion of behavioral descriptions. In addition, every *entity* should have a library statement before its definition, which is “LIBRARY IEEE; USE IEEE.STD_LOGIC_1164.ALL; ”.

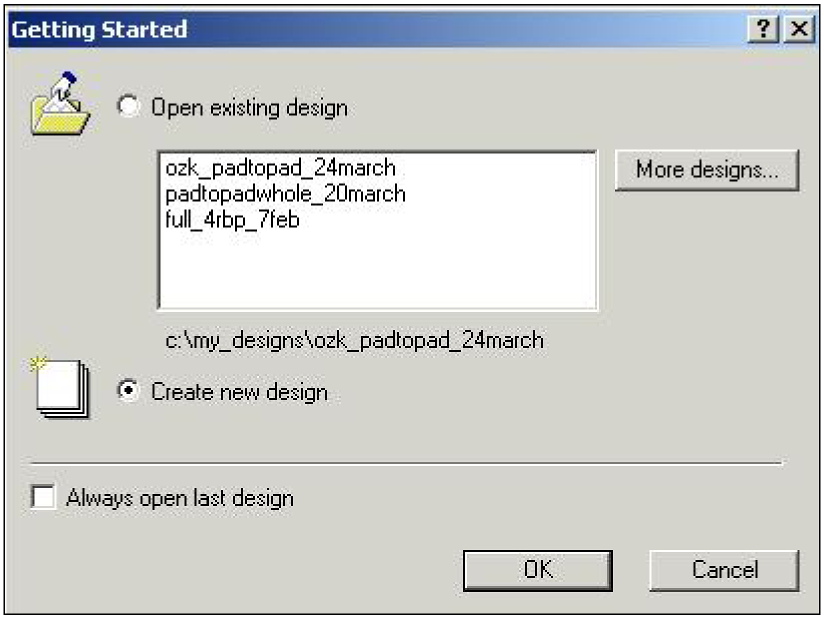
One other thing to be modified in the machine-generated code is the syntax of some entity names. In Active HDL^{TM}, the *entity* or *signal* names cannot contain special characters or special operators. Since the naming convention for *entities* in S_EDIT is not the same as the one in Active HDL^{TM}, there are several names to be changed so they will fit the simulation tool naming rules.

For example, neither “CG_DMSFFPG~CLKreg4_1x” as an *entity* name nor “SELECT” as a *signal* name is accepted in Active HDL^{TM}. The entity name must be changed to “CG_DMSFFPGnotCLKreg4_1x”, while the reserved word SELECT must be modified to “SLCT”. Since the naming corrections are done through the entire code, the modifications do not affect the simulation results and the behavior of the circuit.

4. Example, Inverter

This section contains an example simulation phase of an inverter.

- Open Active HDLTM, by clicking the program icon on the desktop.
- In the dialog box, select “Create New Design” and click “OK” as shown in Figure 20.



Getting Started

○ Open existing design

ozk_padtopad_24march
padtopadwhole_20march
full_4rbp_7feb

c:\my_designs\ozk_padtopad_24march

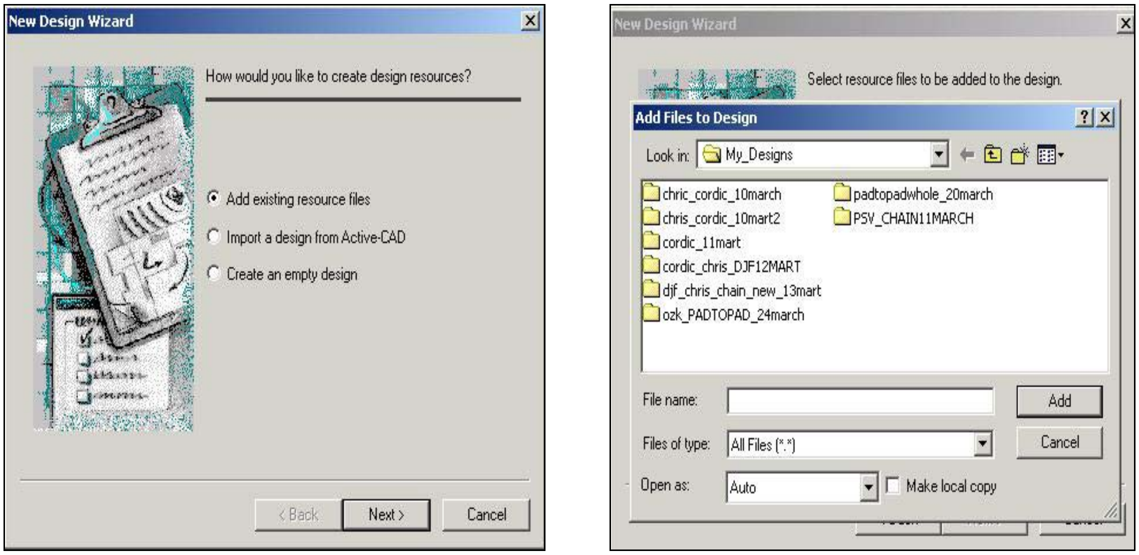
● Create new design

□ Always open last design

Figure 20. Example, Inverter – Creating New Design

- In the dialog box, select “Add Existing Resource Files” and click “Next”. Then, in the second dialog box, select the file generated by S-EDIT with extension “. VHD”. This phase is shown in Figure 21.

- Skip the dialog box for the synthesis tools by clicking “Next” since the design is used only in verification and testing of the circuit. Give a name in the next box for the design and click “Next”. Figure 22 shows this procedure.



New Design Wizard

How would you like to create design resources?

● Add existing resource files
○ Import a design from Active-CAD
○ Create an empty design

< Back | Next > | Cancel

New Design Wizard

Select resource files to be added to the design.

Add Files to Design

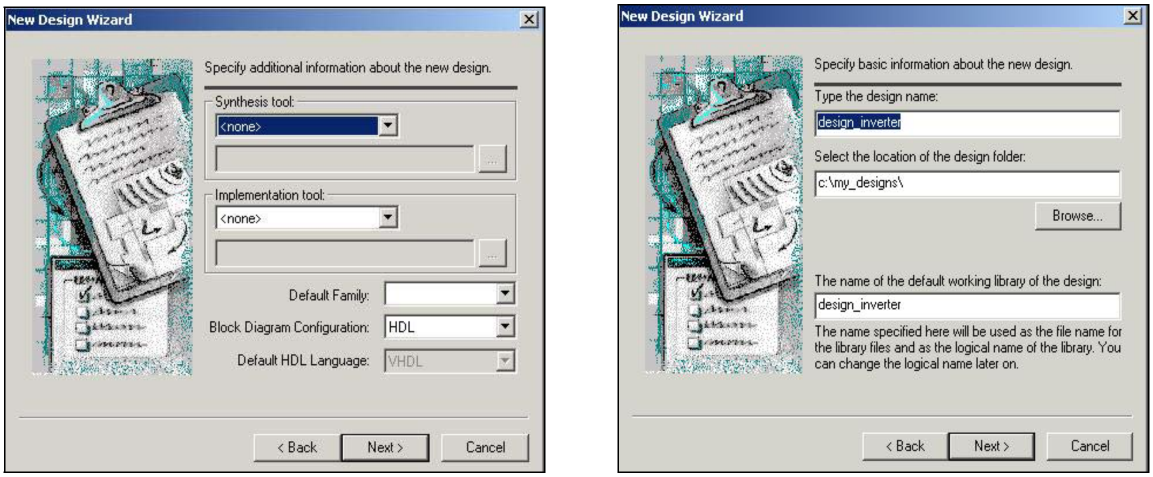
Look in: My_Designs

chric_cordic_10march
chris_cordic_10mart2
cordic_11mart
cordic_chris_DJF12MART
djf_chris_chain_new_13mart
ozk_PADTOPAD_24march
padtopadwhole_20march
PSV_CHAIN11MARCH

File name:
Files of type: All Files (*.*)
Open as: Auto
□ Make local copy

Add | Cancel

Figure 21. Example, Inverter – Adding VHD Code



New Design Wizard

Specify additional information about the new design.

Synthesis tool: <none>
Implementation tool: <none>
Default Family:
Block Diagram Configuration: HDL
Default HDL Language: VHDL

< Back | Next > | Cancel

New Design Wizard

Specify basic information about the new design.

Type the design name: design_inverter

Select the location of the design folder: c:\my_designs\

Browse...

The name of the default working library of the design: design_inverter

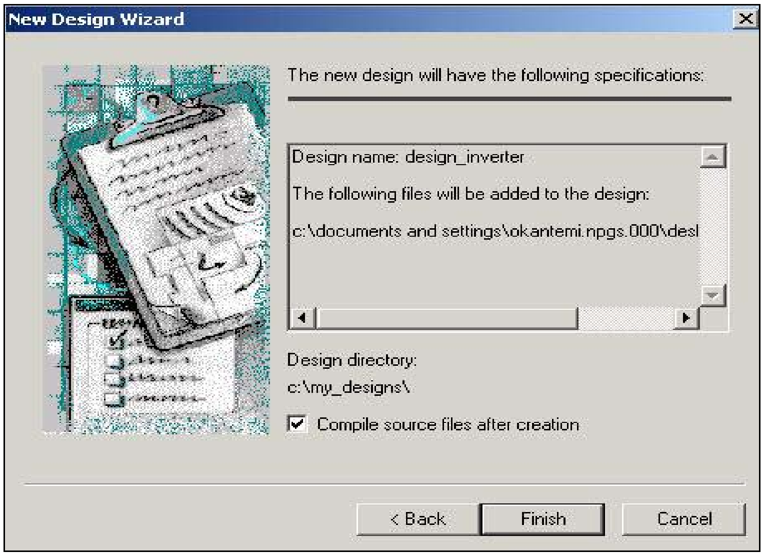
The name specified here will be used as the file name for the library files and as the logical name of the library. You can change the logical name later on.

< Back | Next > | Cancel

Figure 22. Example, Inverter – Naming the Design

- Click “Finish” to Finalize Creating a New Design as shown in Figure 23. Selecting “Compile source files after creation” is not recommended for large VHD codes, since the compilation process may take too much time.

- In the text editor that has appeared, make the modifications necessary and either press “F11” or use Design > Compile menu option/ Shortcut to compile the source file. Repeat this procedure until the code is error and warning free. The corrected code in the text editor is shown in Figure 24.



The new design will have the following specifications:

Design name: design_inverter

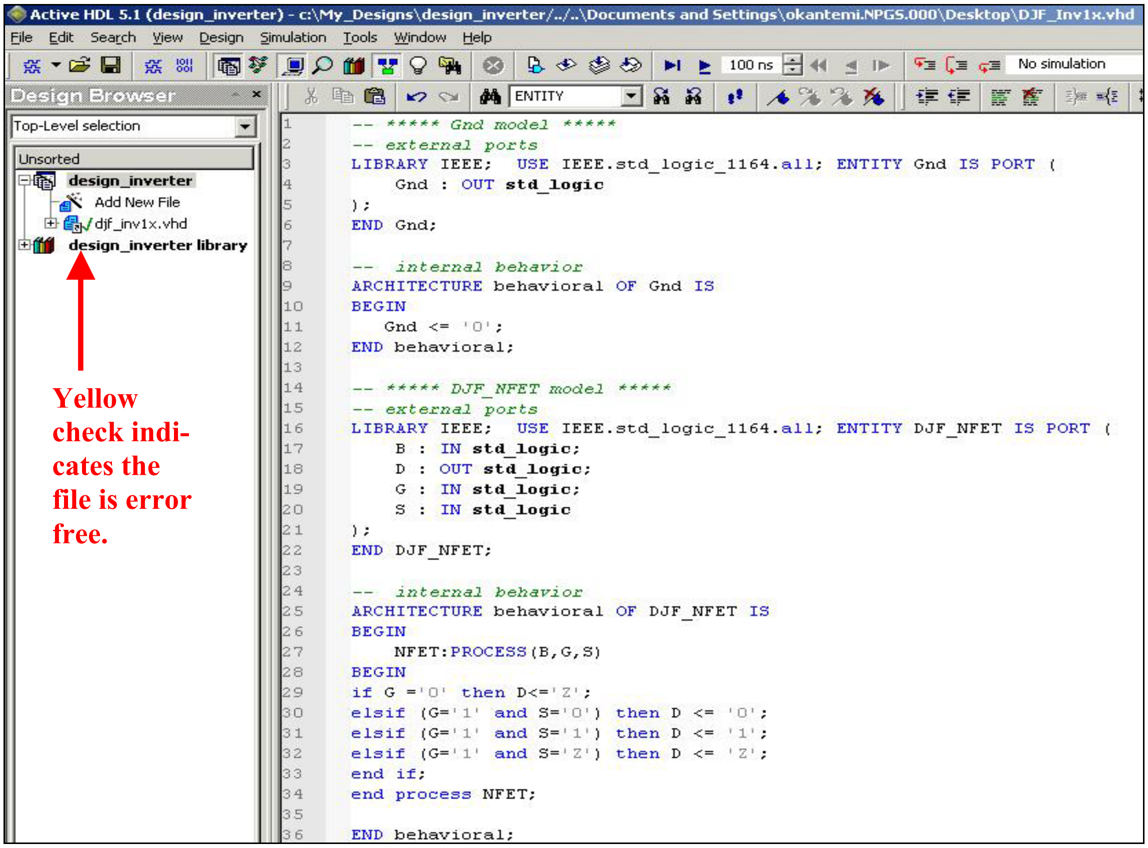
The following files will be added to the design:

c:\documents and settings\okantemi.npgs.000\desl

Design directory:
c:\my_designs\

☑ Compile source files after creation

Figure 23. Example, Inverter – Finishing the New Design Entry

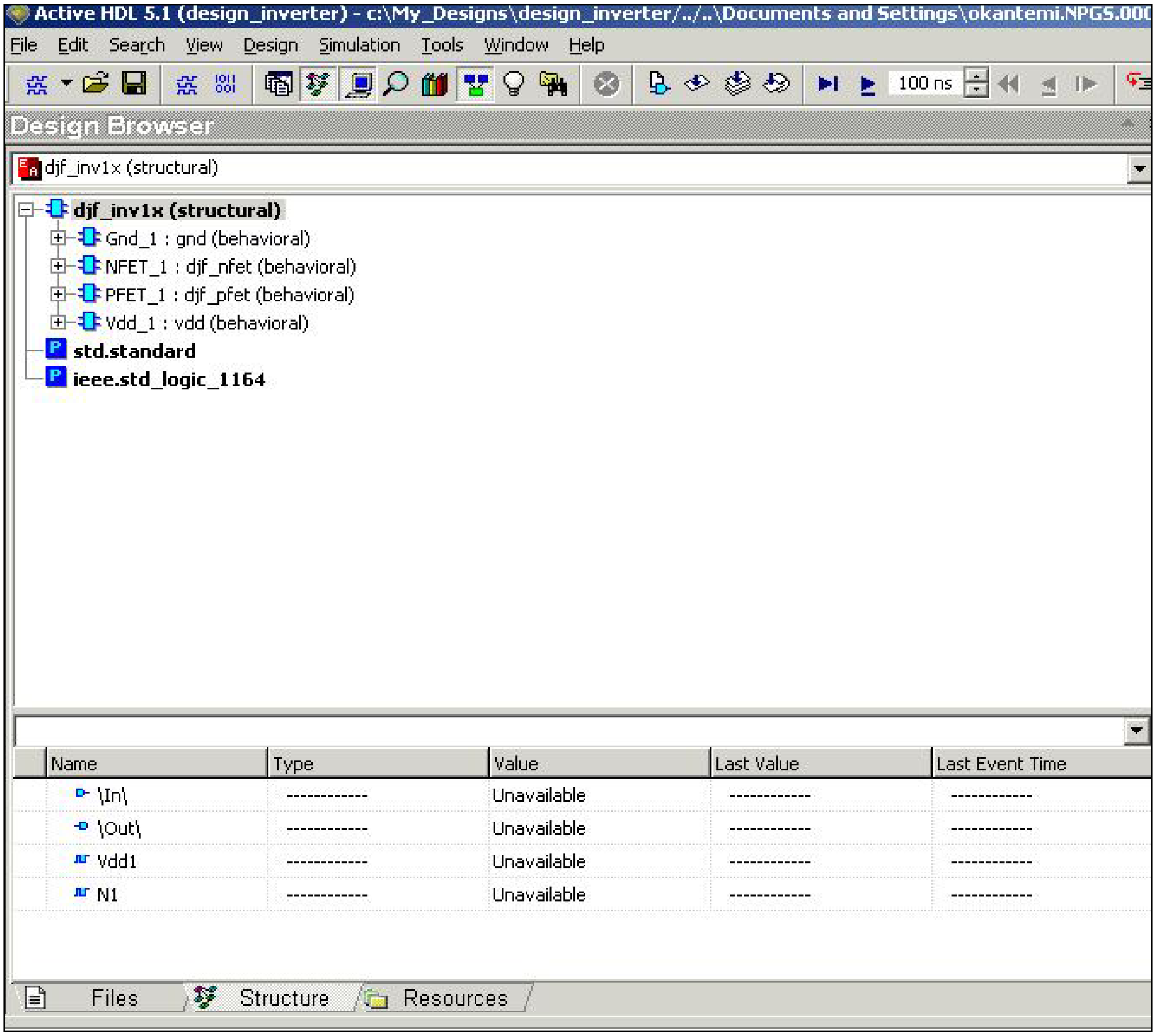


```
-- ***** Gnd model *****
-- external ports
LIBRARY IEEE;  USE IEEE.std_logic_1164.all; ENTITY Gnd IS PORT (
    Gnd : OUT std_logic
);
END Gnd;

--  internal behavior
ARCHITECTURE behavioral OF Gnd IS
BEGIN
   Gnd <= '0';
END behavioral;

-- ***** DJF_NFET model *****
-- external ports
LIBRARY IEEE;  USE IEEE.std_logic_1164.all; ENTITY DJF_NFET IS PORT (
    B : IN std_logic;
    D : OUT std_logic;
    G : IN std_logic;
    S : IN std_logic
);
END DJF_NFET;

--  internal behavior
ARCHITECTURE behavioral OF DJF_NFET IS
BEGIN
    NFET:PROCESS(B,G,S)
BEGIN
if G ='0' then D<='Z';
elsif (G='1' and S='0') then D <= '0';
elsif (G='1' and S='1') then D <= '1';
elsif (G='1' and S='Z') then D <= 'Z';
end if;
end process NFET;

END behavioral;
```

Yellow check indicates the file is error free.

Figure 24. Example, Inverter – Corrected Code

- Select the top-level structure, the inverter, from the roll-down menu in the Design Browser and click on the top level in Structure Section of the Design Browser to see input, output and routing *signals* of interest at that particular *structural* level. Figure 25 shows the Design Browser and Structure Section.


Active HDL 5.1 (design_inverter) - c:\My_Designs\design_inverter/../..\Documents and Settings\okantemi.NPGS.00
File
Edit
Search
View
Design
Simulation
Tools
Window
Help
100 ns
Design Browser
Files
Structure
Resources

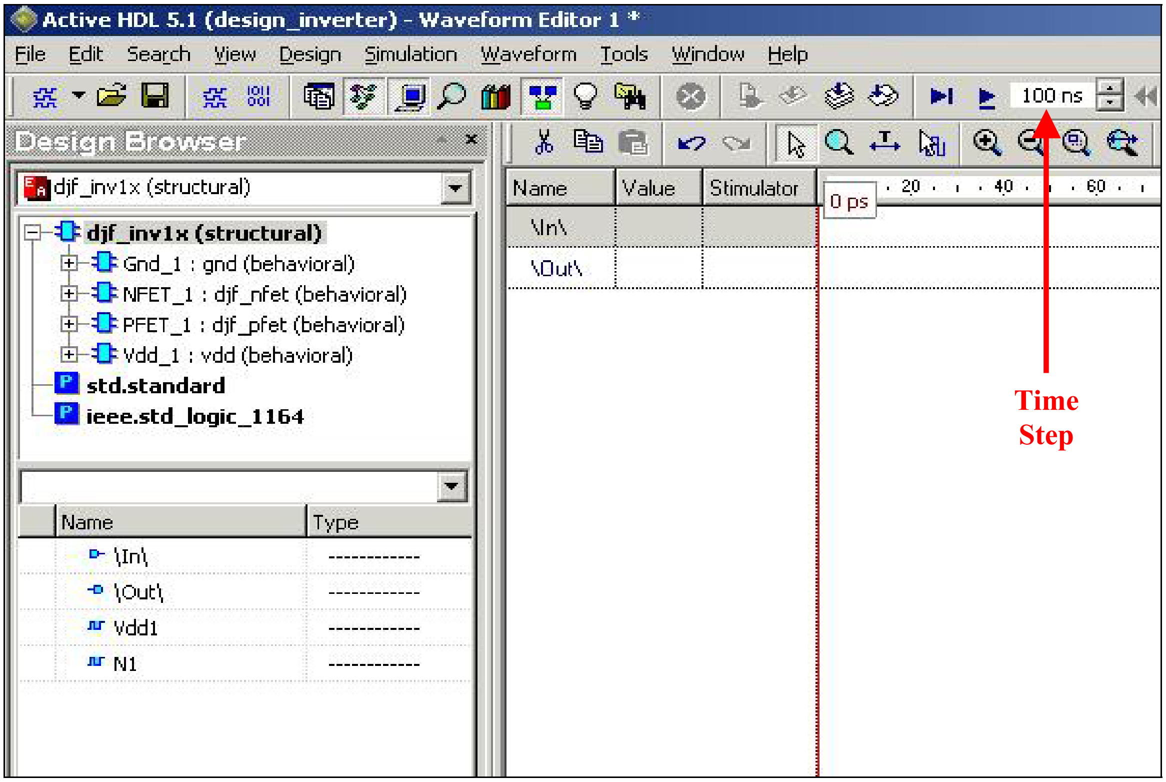
djf_inv1x (structural)

- djf_inv1x (structural)
  - Gnd_1 : gnd (behavioral)
  - NFET_1 : djf_nfet (behavioral)
  - PFET_1 : djf_pfet (behavioral)
  - Vdd_1 : vdd (behavioral)
- std.standard
- ieee.std_logic_1164

Name	Type	Value	Last Value	Last Event Time
\In\	------------	Unavailable	------------	------------
\Out\	------------	Unavailable	------------	------------
Vdd1	------------	Unavailable	------------	------------
N1	------------	Unavailable	------------	------------

Figure 25. Example, Inverter – Top Level Selection

- To generate a waveform, simply select the signals from the signal list by holding the shift key and left clicking on each of the signal names. Right click and select “Add to Waveform” option. The waveform editor generated provides the inputs to be entered in various ways and outputs to be observed in time. Figure 26 shows the Waveform Editor.



Name	Type
\In\	------------
\Out\	------------
Vdd1	------------
N1	------------

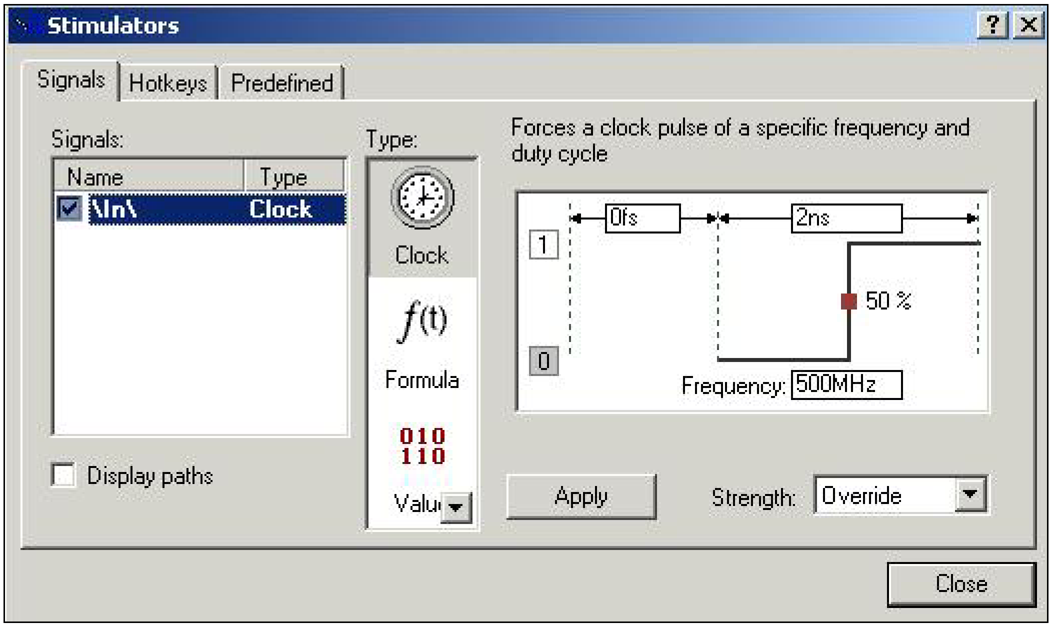
Time Step

Figure 26. Example, Inverter – Waveform Editor

- In the waveform editor, select the input \In\ and right click. Select the “Stimulators” option. The stimulators allow *signals* to be assigned values in various ways. A *signal* can be assigned as a clock, counter, formula or other pre-defined sequences. It is also possible to assign a keyboard button to a signal to toggle the value of that signal in the simulation. For simplicity in this example, select “Clock” option, adjust the frequency and click “Apply”. The Stimulators menu is shown in Figure 27.

- From the Simulation menu select the “Initialize Simulation” option and specify the time step in the box shown in Figure 26. For this example use 10 ns.

- Click once on the right arrow next to the time step box for each simulation step. Repeat as many as necessary to observe the proper input/output relationships. The simulation and the correct operation for the inverter at the end of 10 ns is shown in Figure 28.



Signals:

Name	Type
☑ \In\	Clock

□ Display paths

Type: Clock

Forces a clock pulse of a specific frequency and duty cycle

0fs

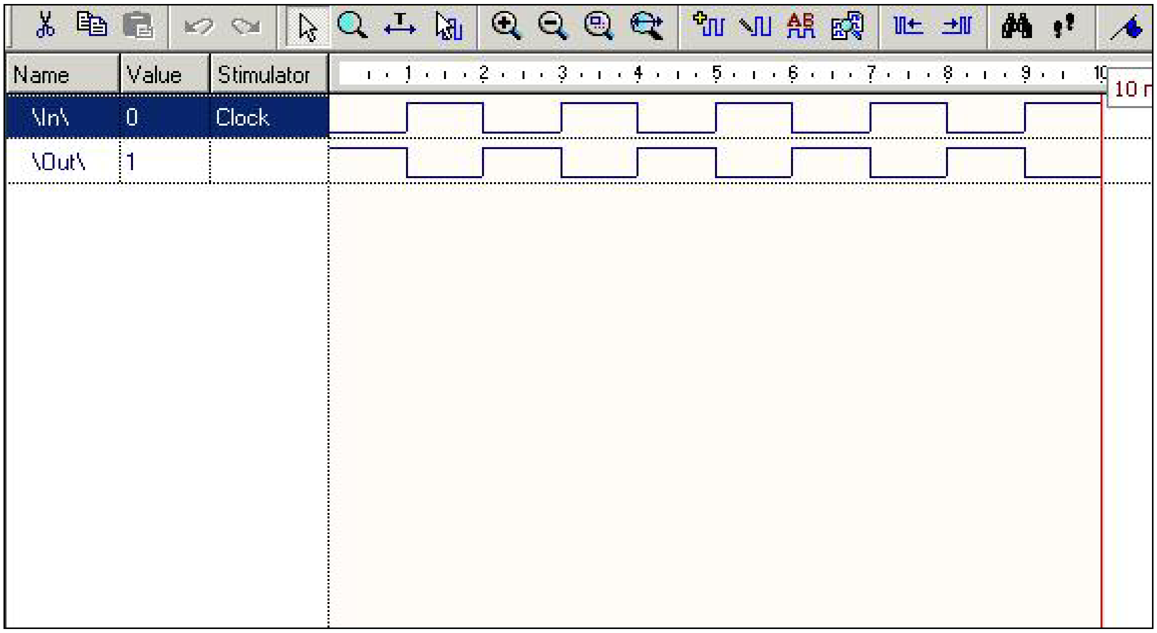
2ns

50 %

Frequency: 500MHz

Strength: Override

Figure 27. Example, Inverter – Stimulators Menu



Name	Value	Stimulator
\In\	0	Clock
\Out\	1	

Figure 28. Example, Inverter – Simulation and Correct Operation

For each code, a block diagram can be generated and used during simulation to trace the signal values in time.

- In order to obtain a block diagram from VHDL code, select the Tools > Code2Graphics Conversion Wizard and follow the instructions in the dialog boxes.

The wizard generates a block diagram for each entity declaration and connects them properly. The graphical representation enhances debugging capabilities and tracing opportunities. Figure 29 presents the generated block diagram and its use in the simulation for the example circuit.

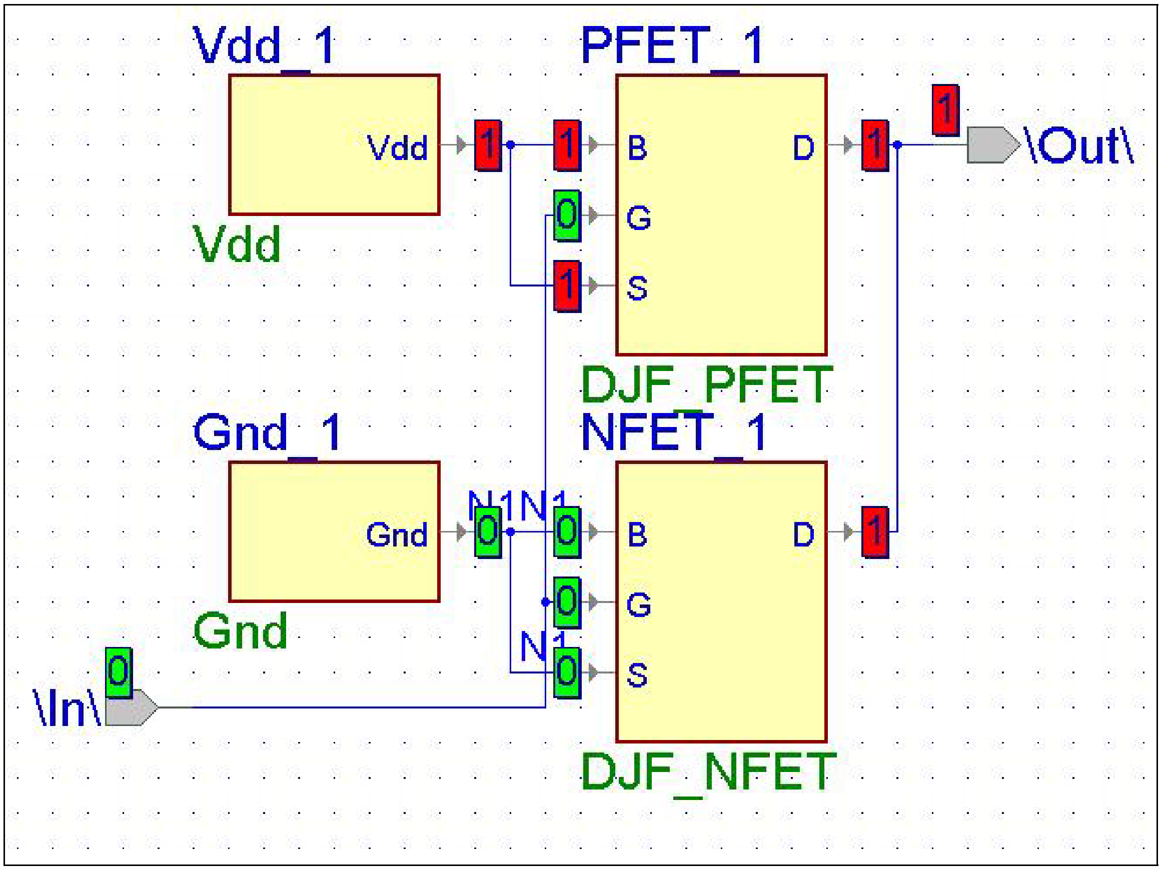
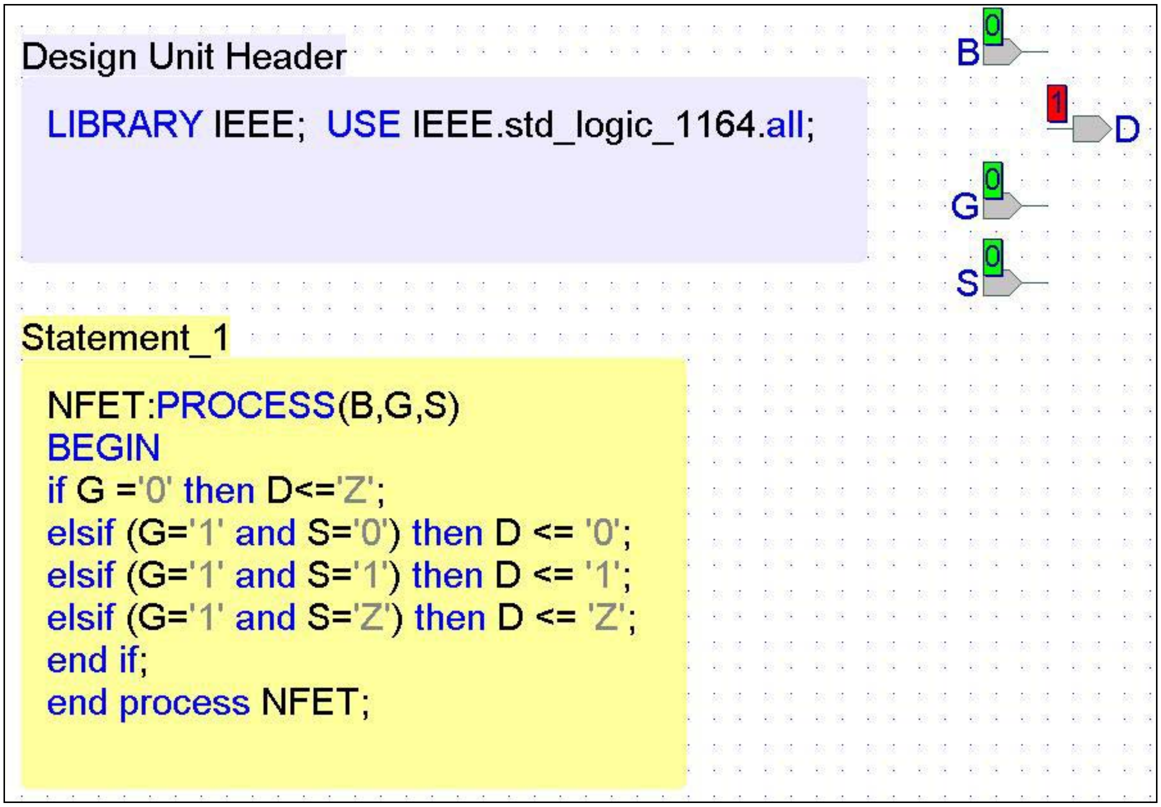


Figure 29. Example, Inverter – Generated Block Diagram for the Top Level

From the top-level graphic, it is possible to navigate down to the lower level *instanced entities* simply by left clicking the box representing the *entity* of interest. An example for the graphical representations of the entities described with behavioral descriptions can be obtained by selecting the *instanced entity* DJF_NFET. The graphical representation of DJF_NFET during simulation is shown in Figure 30.



Design Unit Header

```
LIBRARY IEEE;  USE IEEE.std_logic_1164.all;
```

Statement_1

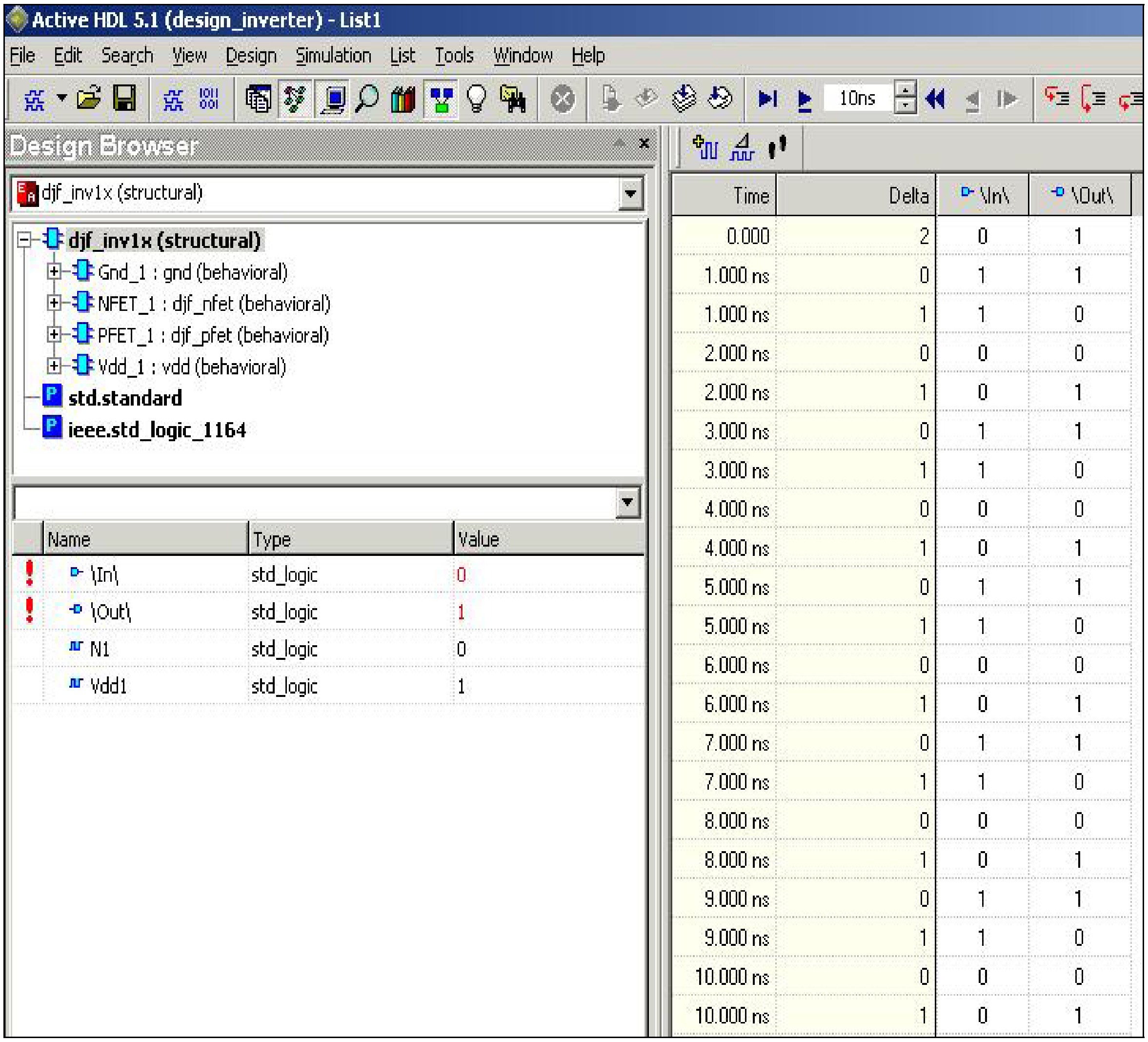
```
NFET:PROCESS(B,G,S)
BEGIN
if G ='0' then D<='Z';
elsif (G='1' and S='0') then D <= '0';
elsif (G='1' and S='1') then D <= '1';
elsif (G='1' and S='Z') then D <= 'Z';
end if;
end process NFET;
```

Figure 30. Example, Inverter –Block Diagram for DJF_NFET

The signal values can be saved as a list by adding them in a List Editor in the same way they can be added in a waveform editor. A list of *signal* values is very helpful in comparing the results with expected values for functional verification. Figure 31 displays a list of the signals for the example inverter in time. This file can be saved as a text file to be processed in any text editor or spreadsheet tool.

5. Reference

The words in italics are protected VHDL constructs. For further reference on Active HDL $^{\text{TM}}$, refer to [14], the vendor firm web site.


Active HDL 5.1 (design_inverter) - List1
File Edit Search View Design Simulation List Tools Window Help
10ns
Design Browser
djf_inv1x (structural)
djf_inv1x (structural)
Gnd_1 : gnd (behavioral)
NFET_1 : djf_nfet (behavioral)
PFET_1 : djf_pfet (behavioral)
Vdd_1 : vdd (behavioral)
std.standard
ieee.std_logic_1164

Name	Type	Value
\In\	std_logic	0
\Out\	std_logic	1
N1	std_logic	0
Vdd1	std_logic	1

Time	Delta	\In\	\Out\
0.000	2	0	1
1.000 ns	0	1	1
1.000 ns	1	1	0
2.000 ns	0	0	0
2.000 ns	1	0	1
3.000 ns	0	1	1
3.000 ns	1	1	0
4.000 ns	0	0	0
4.000 ns	1	0	1
5.000 ns	0	1	1
5.000 ns	1	1	0
6.000 ns	0	0	0
6.000 ns	1	0	1
7.000 ns	0	1	1
7.000 ns	1	1	0
8.000 ns	0	0	0
8.000 ns	1	0	1
9.000 ns	0	1	1
9.000 ns	1	1	0
10.000 ns	0	0	0
10.000 ns	1	0	1

Figure 31. Example, Inverter –List File

This chapter provides an introduction to the VHDL programming language and a guide to the simulation software used to test and verify the DIS. The editors of the Active HDL^{TM} are used extensively in testing the low level cells in the DIS as discussed in Chapter IV.

IV. VHDL SIMULATIONS OF LOW LEVEL CELLS

This chapter shows the simulation methodology of the low level cells used in hardware implementation of the DIS. The components of interest are shown as schematic captures. Input and output signals were introduced. The waveforms or list files used in simulations for verification of each cell are also provided.

A. VERIFICATION OF 5-BIT REGISTER

1. Logic Symbol and Schematic

The logic symbol and circuit schematic for a 5-bit register in S-EDIT are shown in Figure 32 and Figure 33, respectively. For additional information on the design of the circuit, refer to [3] and [12].

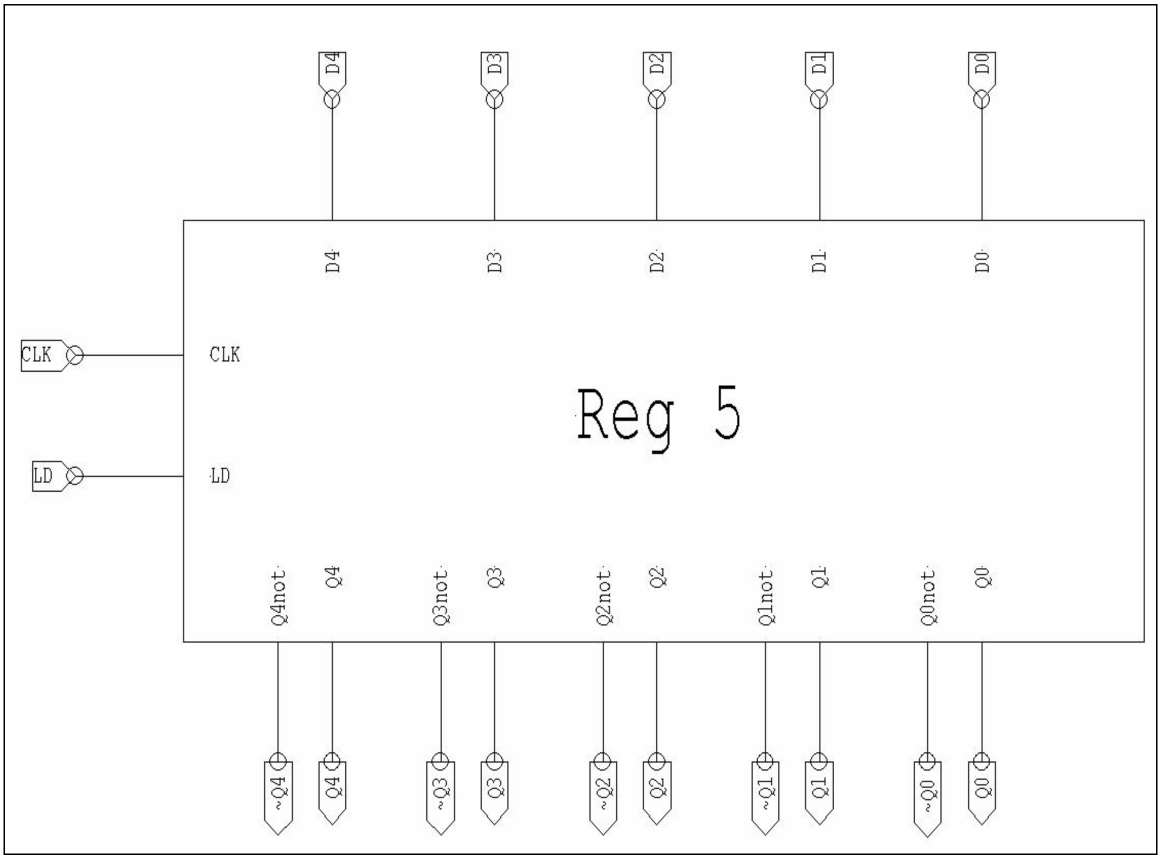


Figure 32. 5-Bit Register Logic Symbol in S-EDIT

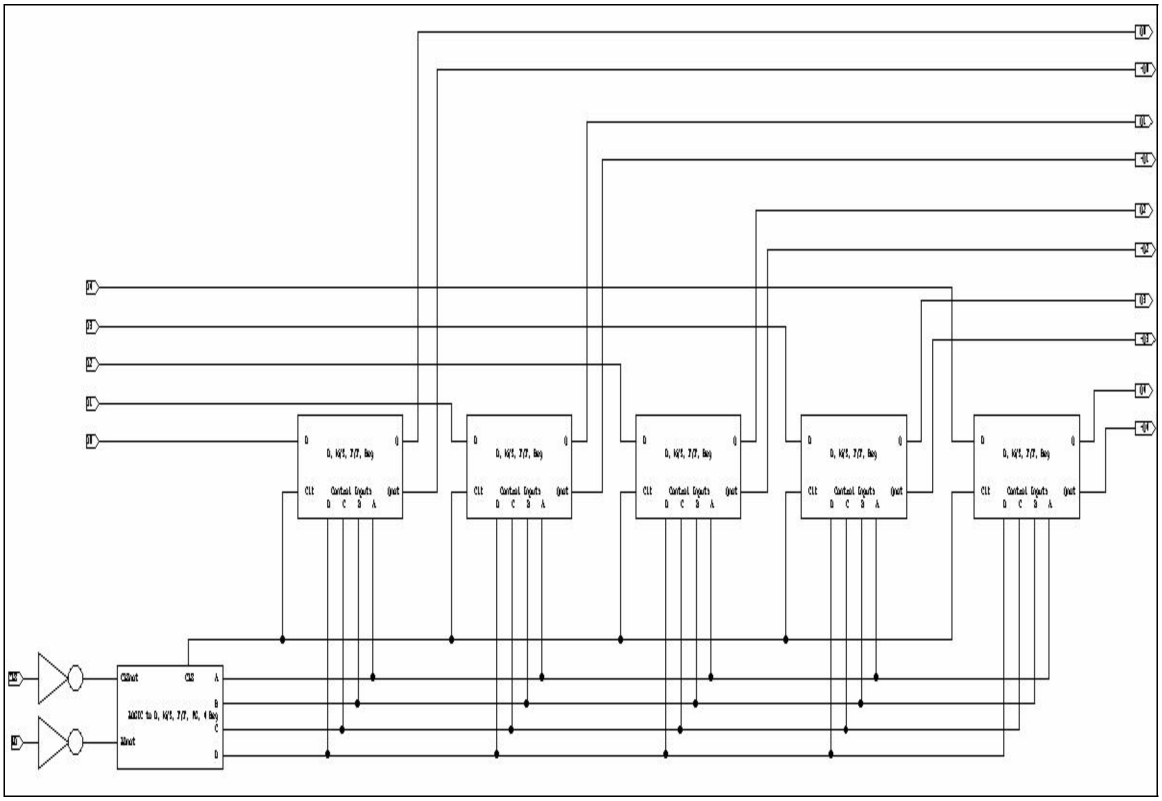


Figure 33. 5-Bit Register Circuit Schematic in S-EDIT

2. Signals

The input and output signals of Figures 32 and 33 are:

- CLK: Clocking signal
- LD: Load signal that latches the inputs into the registers on the rising edge of the clock
- D0 through D4: Input signals
- Q0 through Q4:Output signals that are stored in registers
- ~Q0 through ~Q4: Complements of signals Q0 through Q4.

3. Testing

The state table for the operation of a 1-bit register is given in Table 3. The time at which the inputs are applied is denoted by “t” while the previous value of a signal is represented with “t_0”.

CLK	**LD**	**D**	**Q (t)**	**~Q(t)**
0 to 1	0	0	Q (t_0)	Q (t_0)
0 to 1	0	1	Q (t_0)	Q (t_0)
0 to 1	1	0	0	1
0 to 1	1	1	1	0
1 to 0	0	0	Q (t_0)	Q (t_0)
1 to 0	0	1	Q (t_0)	Q (t_0)
1 to 0	1	0	Q (t_0)	Q (t_0)
1 to 0	1	1	Q (t_0)	Q (t_0)

Table 3. State Table for 1-Bit Register

The methodology explained in Chapter III was used in testing. The VHDL code was used to generate a graphical representation of the circuit in Active HDL^{TM}, which is shown in Figure 34. The waveform used to test the circuit is presented in Figure 35.

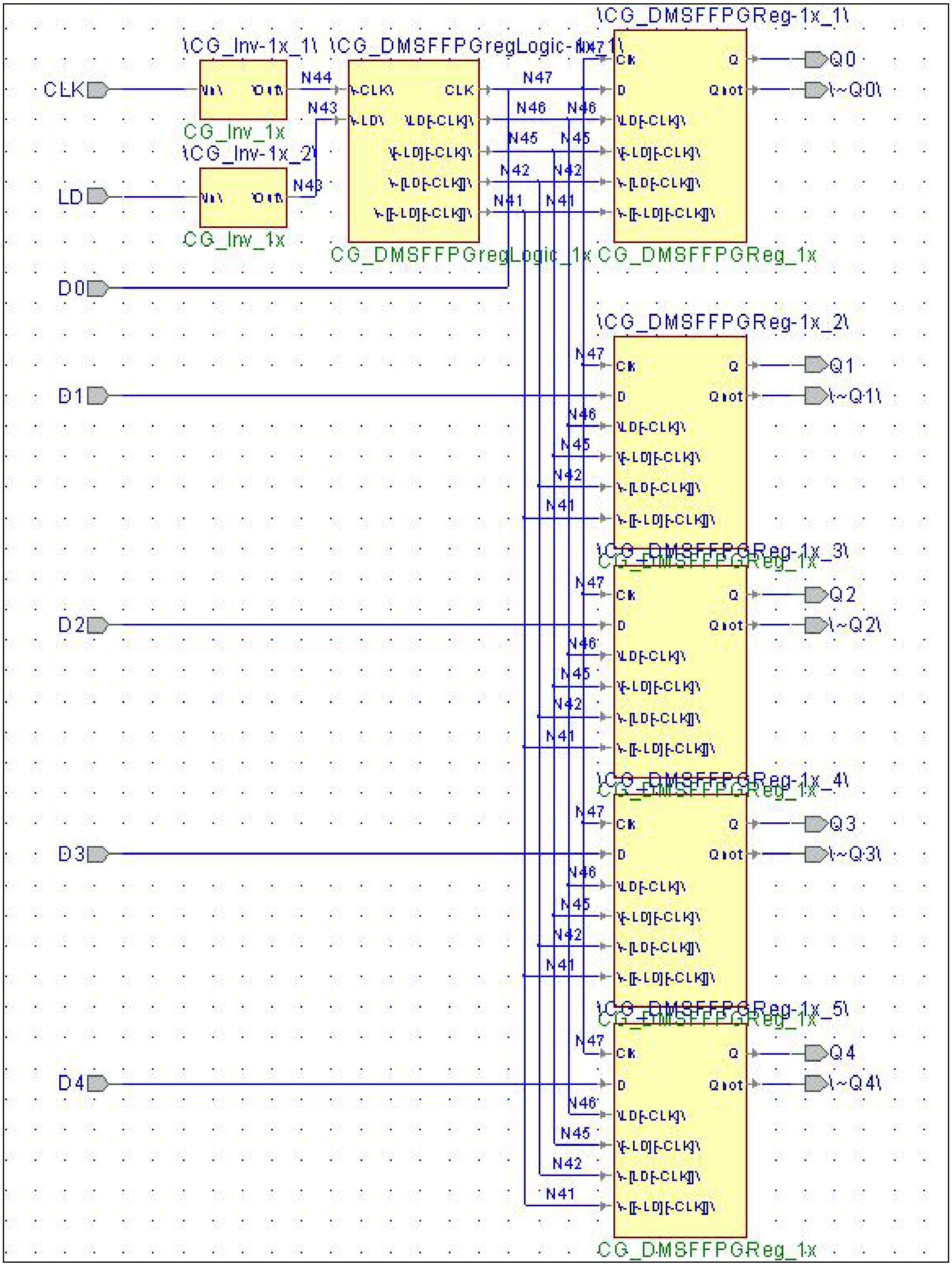
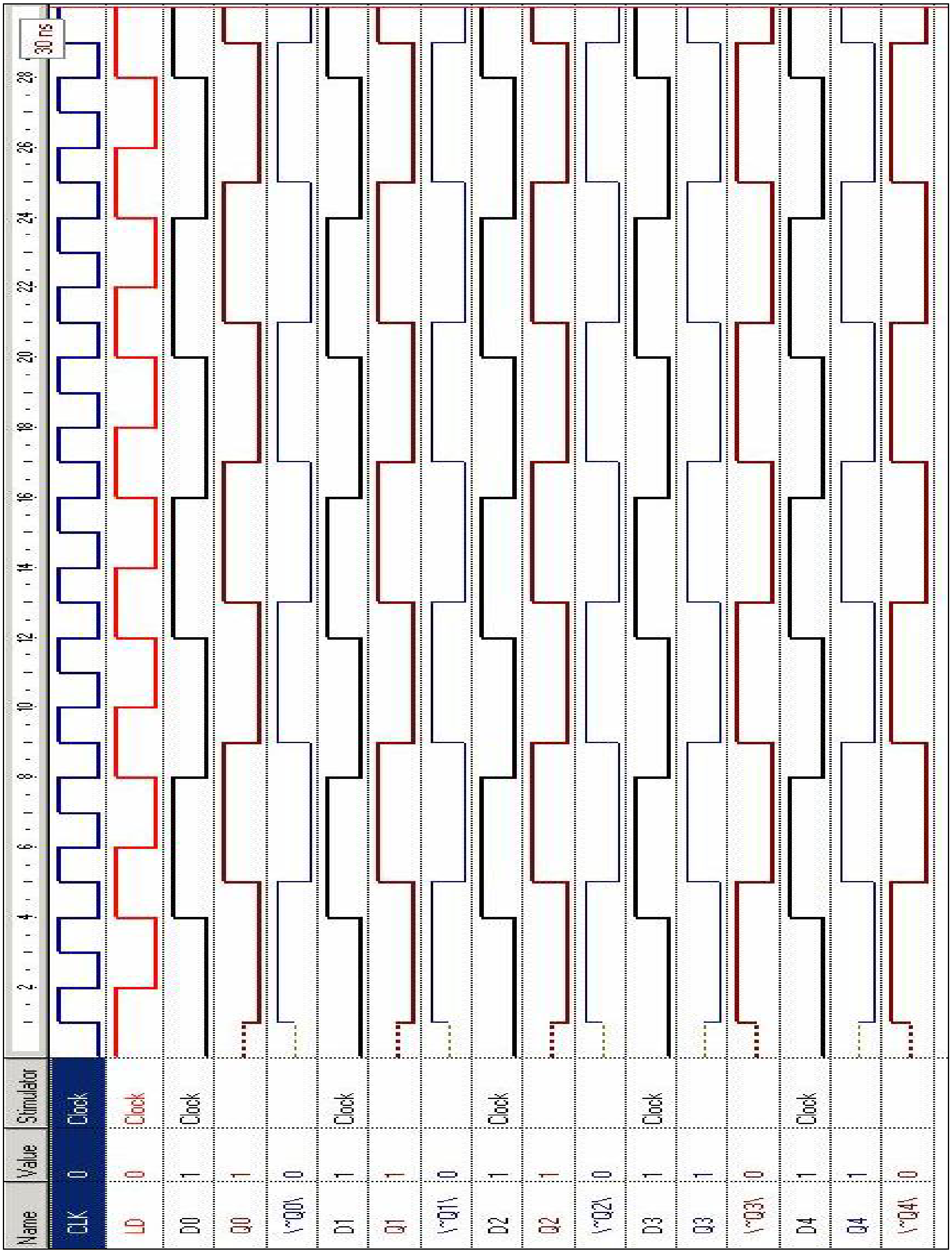


Figure 34. 5-Bit Register Graphical Representation

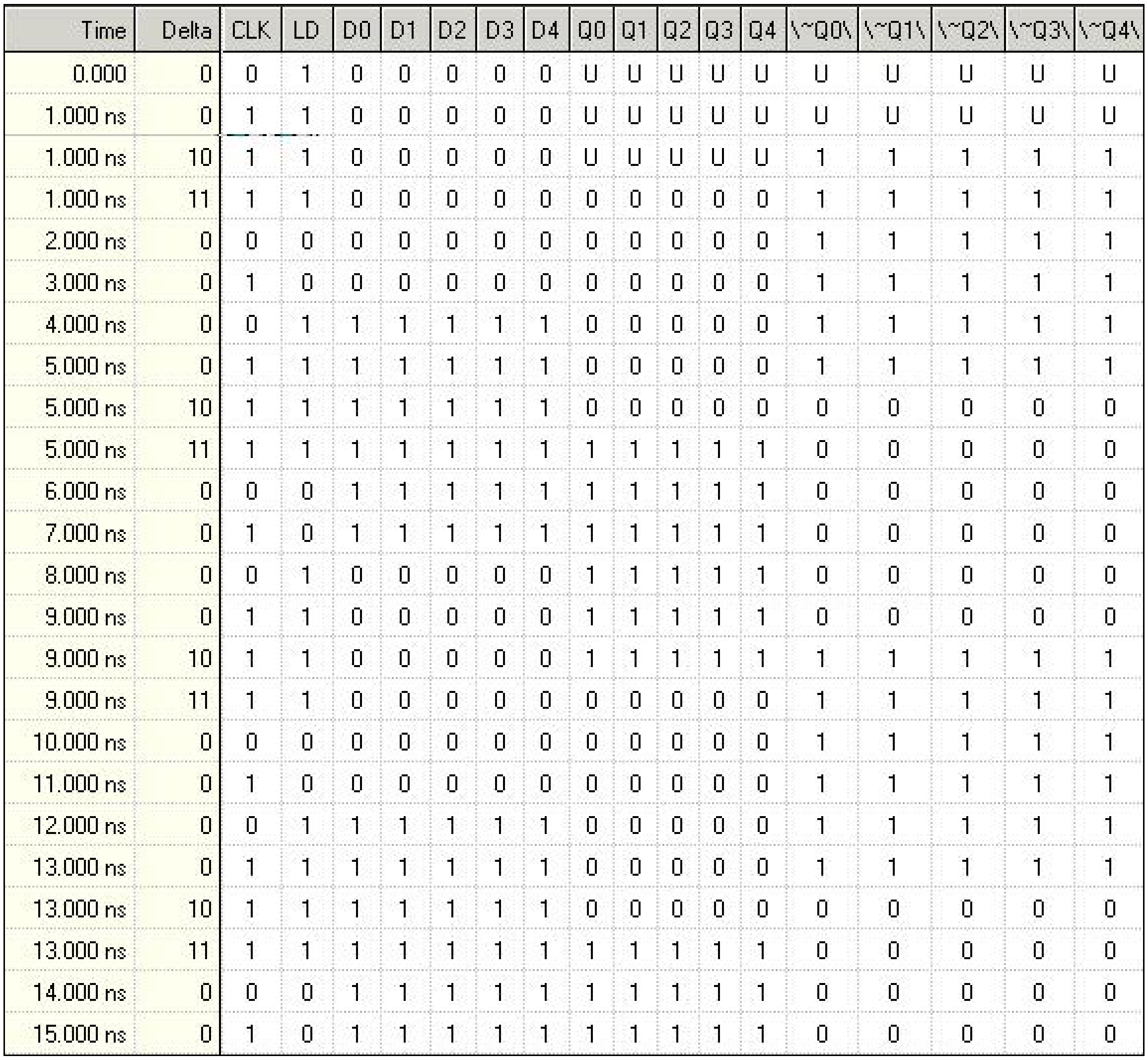


Name	Value	Stimulator
CLK	0	Clock
LD	0	Clock
D0	1	Clock
Q0	1	
\~Q0\	0	
D1	1	Clock
Q1	1	
\~Q1\	0	
D2	1	Clock
Q2	1	
\~Q2\	0	
D3	1	Clock
Q3	1	
\~Q3\	0	
D4	1	Clock
Q4	1	
\~Q4\	0	

Figure 35. Waveform Showing Proper Operation for 5-Bit Register

4. Verification

The List File, shown in Figure 36, obtained from the simulation shows the values of the signals with respect to time.



Time	Delta	CLK	LD	D0	D1	D2	D3	D4	Q0	Q1	Q2	Q3	Q4	\~Q0\	\~Q1\	\~Q2\	\~Q3\	\~Q4\
0.000	0	0	1	0	0	0	0	0	U	U	U	U	U	U	U	U	U	U
1.000 ns	0	1	1	0	0	0	0	0	U	U	U	U	U	U	U	U	U	U
1.000 ns	10	1	1	0	0	0	0	0	U	U	U	U	U	1	1	1	1	1
1.000 ns	11	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
2.000 ns	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
3.000 ns	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
4.000 ns	0	0	1	1	1	1	1	1	0	0	0	0	0	1	1	1	1	1
5.000 ns	0	1	1	1	1	1	1	1	0	0	0	0	0	1	1	1	1	1
5.000 ns	10	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
5.000 ns	11	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
6.000 ns	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
7.000 ns	0	1	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
8.000 ns	0	0	1	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0
9.000 ns	0	1	1	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0
9.000 ns	10	1	1	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
9.000 ns	11	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
10.000 ns	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
11.000 ns	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
12.000 ns	0	0	1	1	1	1	1	1	0	0	0	0	0	1	1	1	1	1
13.000 ns	0	1	1	1	1	1	1	1	0	0	0	0	0	1	1	1	1	1
13.000 ns	10	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
13.000 ns	11	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
14.000 ns	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
15.000 ns	0	1	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0

Figure 36. List Editor Showing Proper Operation for 5-Bit Register

It can be seen that the output transitions occurred at the low-to-high change of the clock signal as long as the input LD is high. Counters were used as input signals. It is important to keep the LD and D inputs stable during low-to-high clock transition to observe proper circuit behavior.

Table 4 compares the simulation results and state table for the circuit. By reviewing the data in Table 4 and comparing the columns labeled "Simulation Results" against the columns labeled "State Table", it can be seen that the simulation ran correctly.

Control Inputs		**State Table**			**Simulation Results**		
CLK	**LD**	**D**	**Q**	**~Q**	**D**	**Q**	**~Q**
0 to 1	0	0	$D(t_0)$	$\sim D(t_0)$	0	$D(t_0)$	$\sim D(t_0)$
0 to 1	0	1	$D(t_0)$	$\sim D(t_0)$	1	$D(t_0)$	$\sim D(t_0)$
0 to 1	1	0	0	1	0	0	1
0 to 1	1	1	1	0	1	1	0
1 to 0	0	0	$D(t_0)$	$\sim D(t_0)$	0	$D(t_0)$	$\sim D(t_0)$
1 to 0	0	1	$D(t_0)$	$\sim D(t_0)$	1	$D(t_0)$	$\sim D(t_0)$
1 to 0	1	0	$D(t_0)$	$\sim D(t_0)$	0	$D(t_0)$	$\sim D(t_0)$
1 to 0	1	1	$D(t_0)$	$\sim D(t_0)$	1	$D(t_0)$	$\sim D(t_0)$

Table 4. Comparing Simulation Results and State Table for 5-Bit Register

B. VERIFICATION OF ~S/~R LATCH

1. Logic Symbol and Schematic

The logic symbol and circuit schematic for an ~S/~R Latch in S-EDIT are shown in Figures 37 and 38, respectively.

Buffers in Figure 38 are added to the schematic before extracting the VHDL code to accommodate a rule in Active HDL$^{\text{TM}}$ requiring users to avoid networks that go to a logic gate input and an output port. Furthermore, in order to avoid assigning an output signal as an input signal for the circuit itself, a behavioral description for the latch was inserted in the VHDL code, which is given in Figure 39.

2. Signals

- ~S/~R: Complement of Set/Reset signals in a regular S/R latch.
- Q/QN: Stored latch value at time "t".

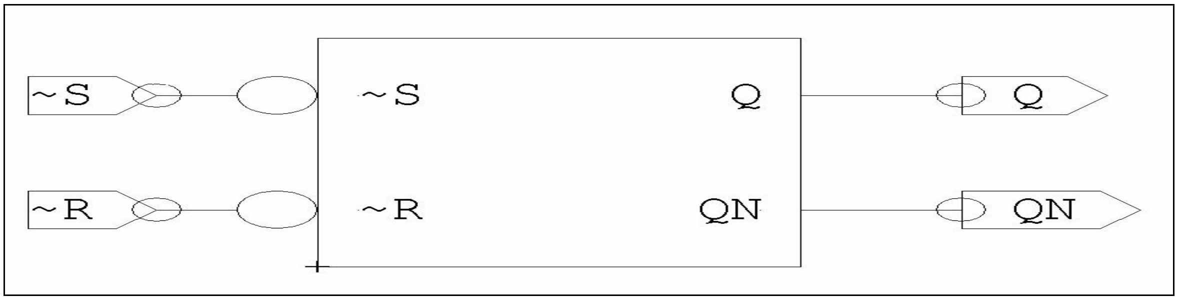


Figure 37. ~S/~R Latch Logic Symbol in S-EDIT

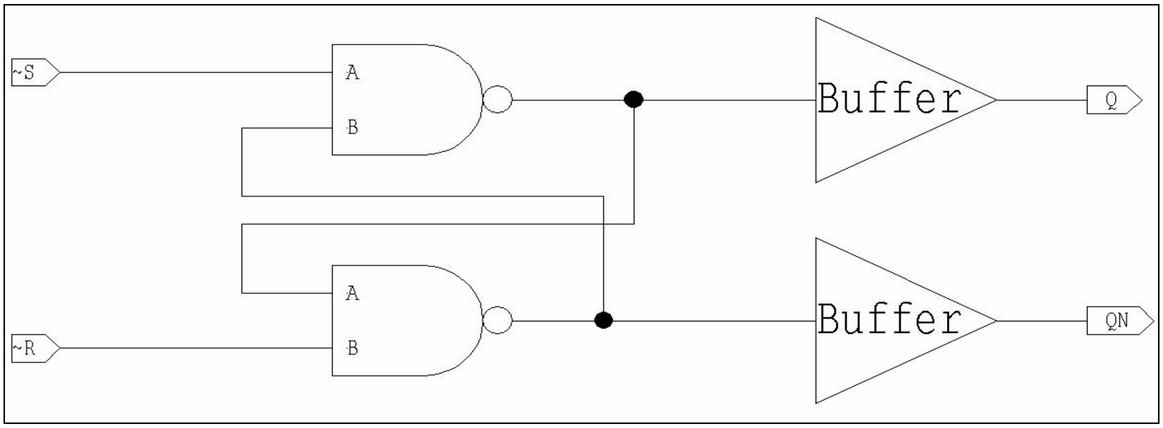


Figure 38. ~S/~R Latch Circuit Schematic in S-EDIT

3. Testing

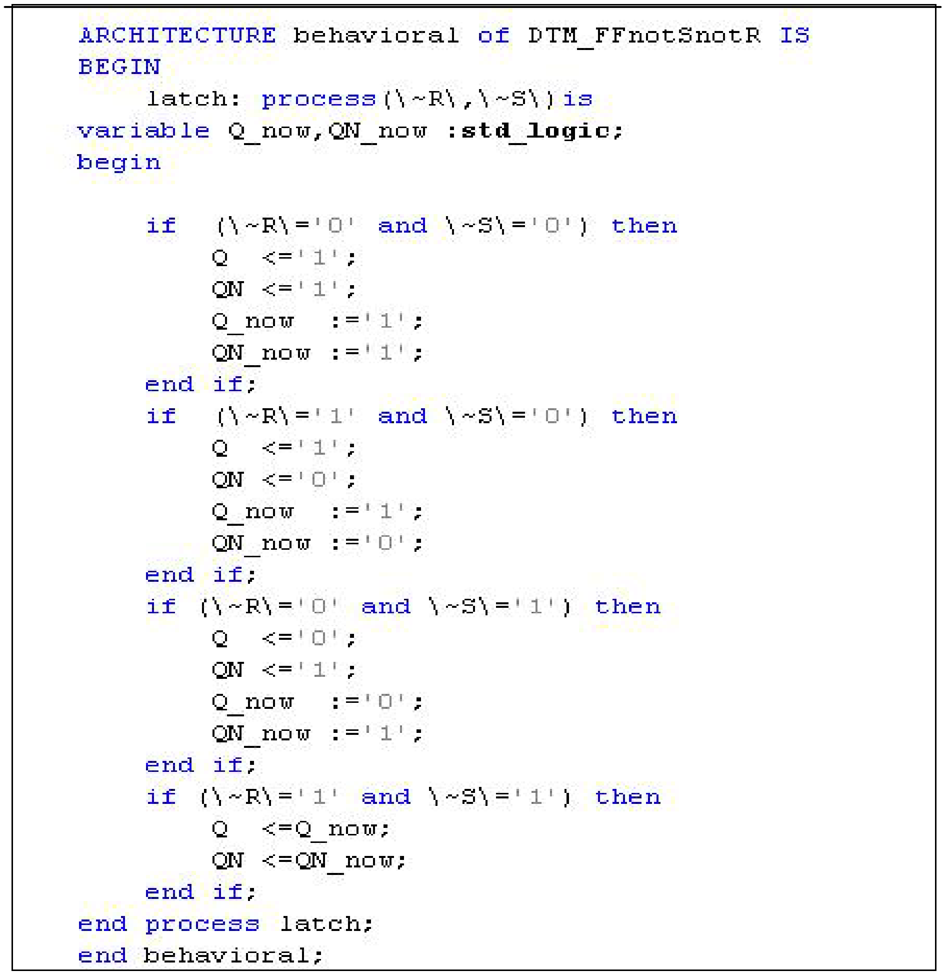
The state table for the operation of the ~S/~R is given in Table 5. The present input values are denoted by “t” while the previous value of a signal is represented with “t_0”.

~S	**~R**	**Q (t)**	**QN (t)**
1	1	Q (t_0)	QN (t_0)
1	0	0	1
0	1	1	0
0	0	Not Allowed	

Table 5. State Table for ~S/~R Latch

The behavioral description implements the state table by using two different variables for Q_now and QN_now to define the latch state for initialization purposes. Since the signal values are computed in a single simulation cycle, assigning unknown values to any input would result in unknown states at the outputs, which in turn would cause an infinite loop resulting in incorrect simulation results.

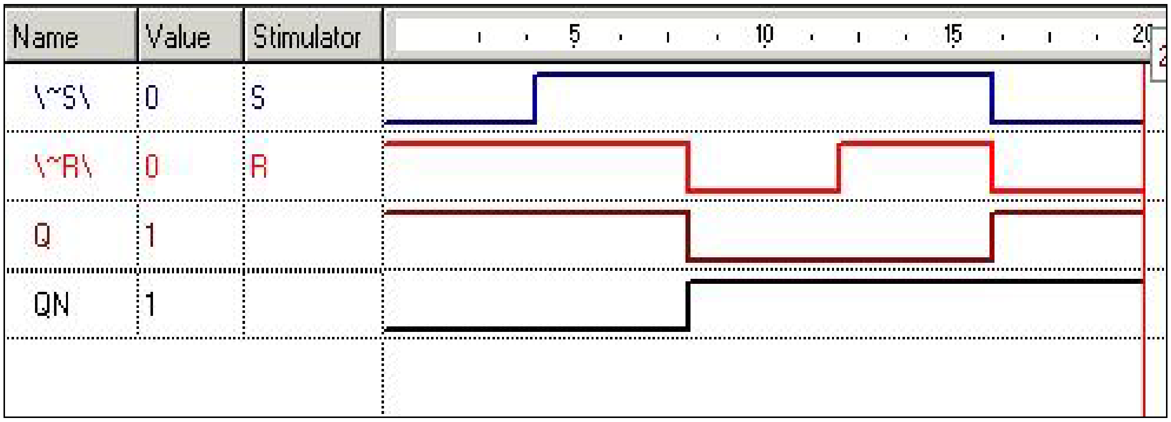
Although the (~S, ~R)=(0,0) case is not allowed in the state table, for initialization purposes, this set of inputs are included in the behavioral description.



```
ARCHITECTURE behavioral of DTM_FFnotSnotR IS
BEGIN
    latch: process(\~R\,\~S\)is
variable Q_now,QN_now :std_logic;
begin

    if  (\~R\='0' and \~S\='0') then
        Q  <='1';
        QN <='1';
        Q_now  :='1';
        QN_now :='1';
    end if;
    if  (\~R\='1' and \~S\='0') then
        Q  <='1';
        QN <='0';
        Q_now  :='1';
        QN_now :='0';
    end if;
    if (\~R\='0' and \~S\='1') then
        Q  <='0';
        QN <='1';
        Q_now  :='0';
        QN_now :='1';
    end if;
    if (\~R\='1' and \~S\='1') then
        Q  <=Q_now;
        QN <=QN_now;
    end if;
end process latch;
end behavioral;
```

Figure 39. Behavioral Description of ~S/~R Latch

The waveform used in testing the circuit is shown in Figure 40. The state of the latch is initialized to the Q, QN)=(1,1) case in the behavioral description since these values make the NAND gates sensitive to the ~S and ~R inputs.

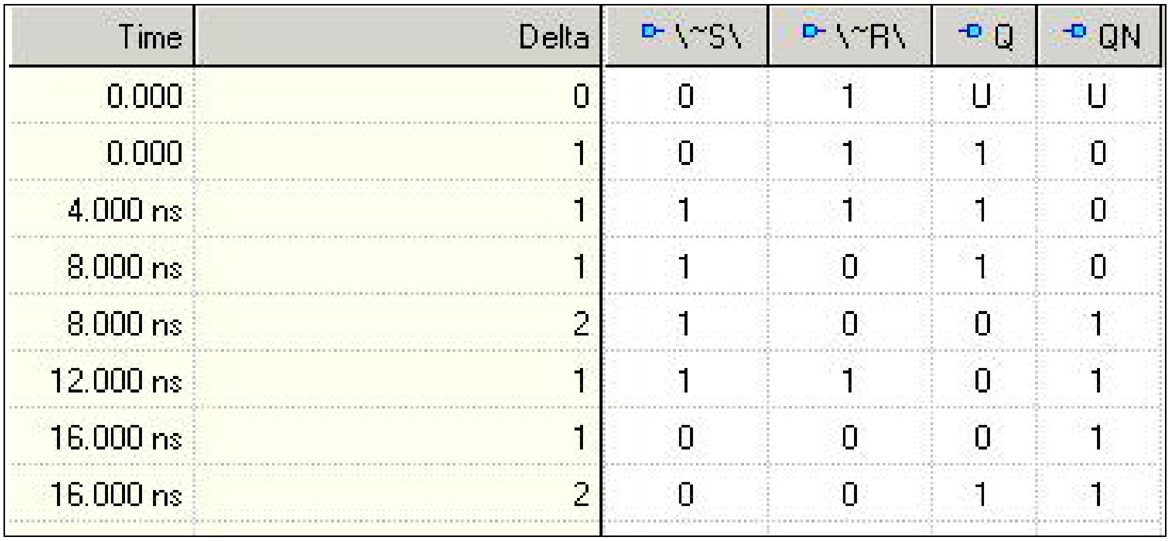


Name	Value	Stimulator
\~S\	0	S
\~R\	0	R
Q	1	
QN	1	

Figure 40. Waveform Showing Proper Operation for ~S/~R Latch

4. Verification

The List File is given in Figure 41, while Table 6 compares the simulation results, and the state table.



Time	Delta	\~S\	\~R\	Q	QN
0.000	0	0	1	U	U
0.000	1	0	1	1	0
4.000 ns	1	1	1	1	0
8.000 ns	1	1	0	1	0
8.000 ns	2	1	0	0	1
12.000 ns	1	1	1	0	1
16.000 ns	1	0	0	0	1
16.000 ns	2	0	0	1	1

Figure 41. List Editor Showing Proper Operation for ~S/~R Latch

Inputs		State Table		Simulation Results	
~S	~R	Q	QN	Q	QN
0	1	1	0	1	0
1	1	1	0	1	0
1	0	0	1	0	1
1	1	0	1	0	1
0	0	Not Allowed		1	1

Table 6. Comparing Simulation Results and State Table for ~S/~R Latch

C. VERIFICATION OF 12-BIT COMPARATOR

1. Logic Symbol and Schematic

A 12-bit comparator is used in the overhead circuitry as a part of the self-test mechanism. It compares the number of test vectors generated, which was supplied by a binary counter, and the desired test vector number provided externally. If the two numbers are the same, the output signal causes the cascade of Range Bin Processors (RBP s) to switch into the Maintenance Mode. The correct operation of the comparator in combination with the counter is of great importance to the self-test logic. The comparator’s logic symbol and schematic are given in Figures 42 and 43, respectively.

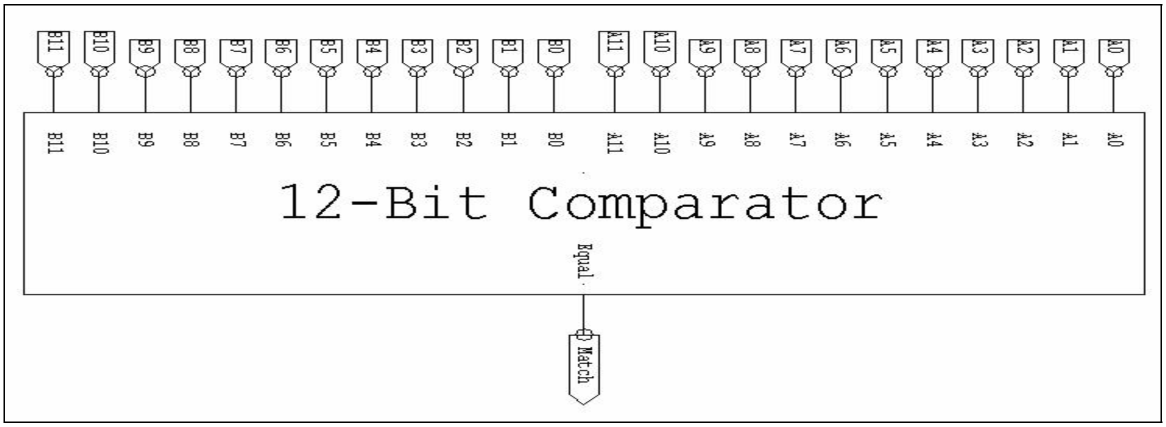


Figure 42. 12-Bit Comparator Logic Symbol in S-EDIT

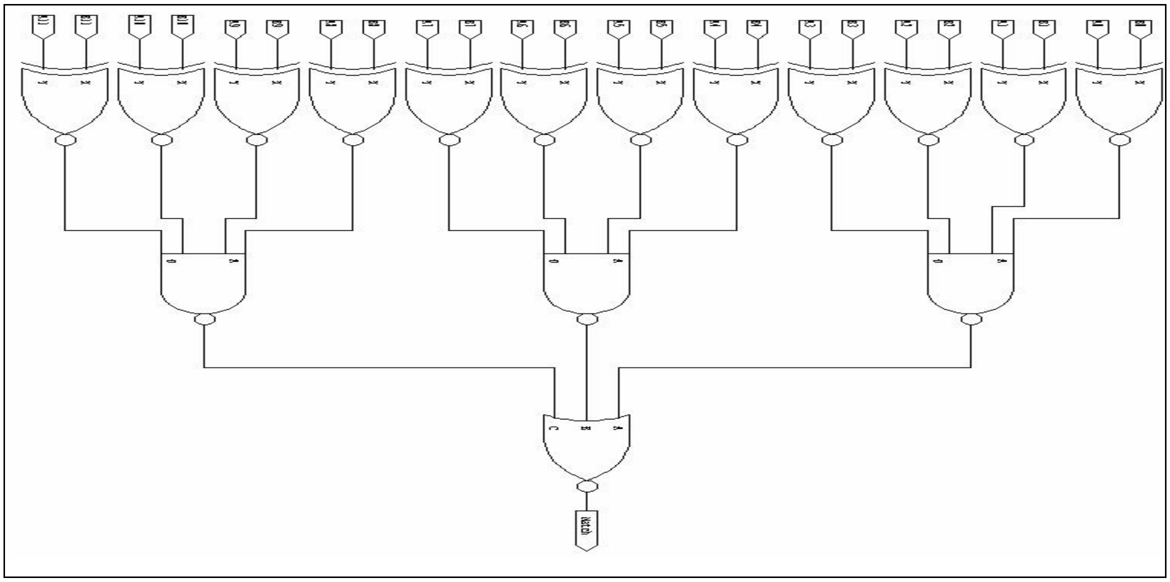


Figure 43. 12-Bit Comparator Circuit Schematic in S-EDIT

2. Signals

The input and output signals of Figures 42 and 43 are:

- A0 through A11: Inputs from the counter
- B0 through B11: Off-Chip Count inputs
- Equal: Output signal effecting Operate/Maintenance input to the RBP s via an ~S/~R latch.

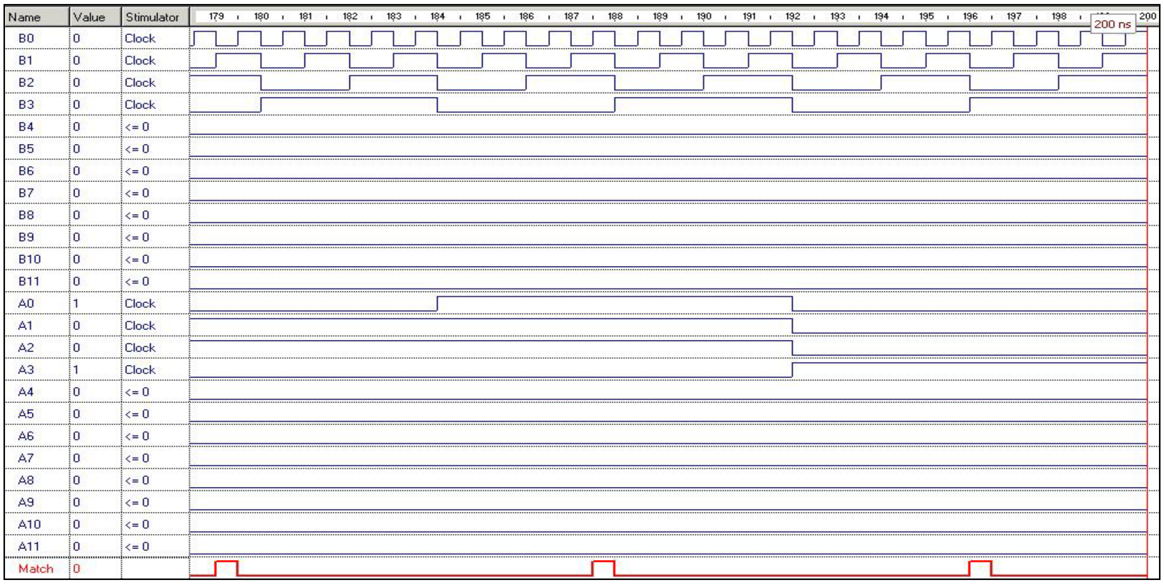
3. Testing

The truth table for the comparator is given in Table 7.

A	**B**	**Match**
A ≠ B		0
A = B		1

Table 7. Truth Table for 12-Bit Comparator

By using every possible value for the A and B signals, and by observing the output Match signal, an exhaustive test was conducted. A part of the waveform generated is shown in Figure 44. The graphical representation generated in Active HDL^{TM} is given in Figure 45.



Name	Value	Stimulator
B0	0	Clock
B1	0	Clock
B2	0	Clock
B3	0	Clock
B4	0	<= 0
B5	0	<= 0
B6	0	<= 0
B7	0	<= 0
B8	0	<= 0
B9	0	<= 0
B10	0	<= 0
B11	0	<= 0
A0	1	Clock
A1	0	Clock
A2	0	Clock
A3	1	Clock
A4	0	<= 0
A5	0	<= 0
A6	0	<= 0
A7	0	<= 0
A8	0	<= 0
A9	0	<= 0
A10	0	<= 0
A11	0	<= 0
Match	0	

Figure 44. Waveform Showing Proper Operation for 12-Bit Comparator

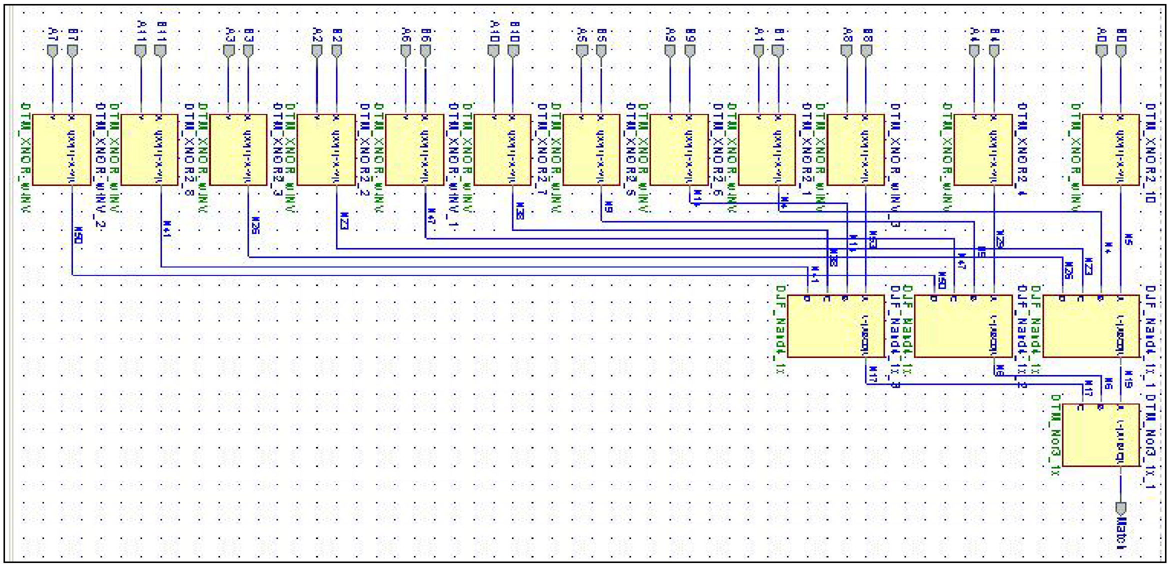


Figure 45. 12-Bit Comparator Graphical Representation

4. Verification

Results are verified by comparing the List File for the simulation for the A and B vectors and the resultant Match signal with a separate Matlab generated text file. The results perfectly match with the state table. The algorithm used to generate the Matlab code and set up the input values in Active HDL is presented in Table 8.

MATLAB	ACTIVE HDL^{TM} Waveform Editor Signal Stimulator Clock Periods							

for every_value of vector_A
begin
for every_value of Vector B
begin
if Value_A=Value_B then
Match=1;
Else Match=0;
end;
end;
	A0	2^0	A6	2^6	B0	2^12	B6	2^18

	A1	2^1	A7	2^7	B1	2^13	B7	2^19
	A2	2^2	A8	2^8	B2	2^14	B8	2^20
	A3	2^3	A9	2^9	B3	2^15	B9	2^21
	A4	2^4	A10	2^10	B4	2^16	B10	2^22
	A5	2^5	A11	2^11	B5	2^17	B11	2^23

Table 8. Exhaustive Test and Verification Algorithm for 12-Bit Comparator

D. VERIFICATION OF 5-BIT ADDER

1. Logic Symbol and Schematic

Different types of adders are used in the Digital Image Synthesizer (DIS). Their proper operation is of great importance for correct target signature generation. Here, a 5-Bit Adder, whose logic symbol/schematic are provided in Figures 46 and 47, is tested for proper operation. For further information on the carry look-ahead adder design, refer to [3].

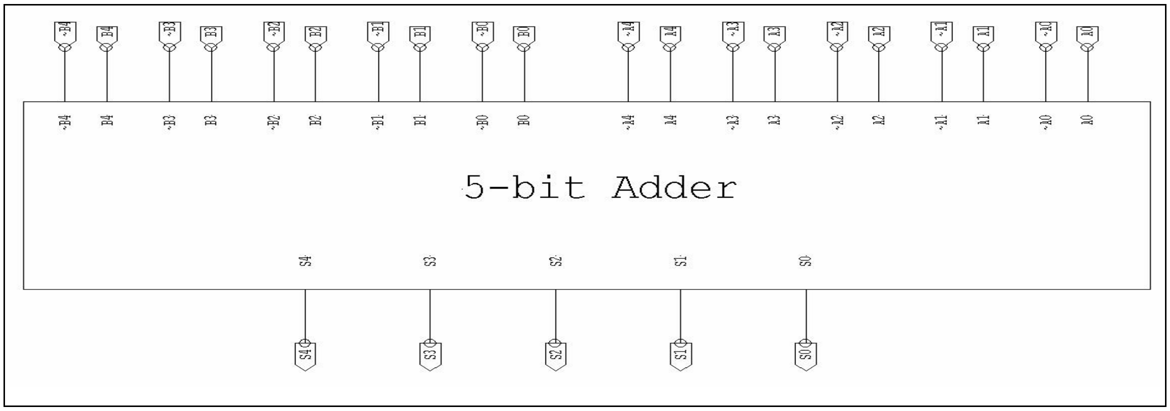


Figure 46. 5-Bit Adder Logic Symbol in S-EDIT

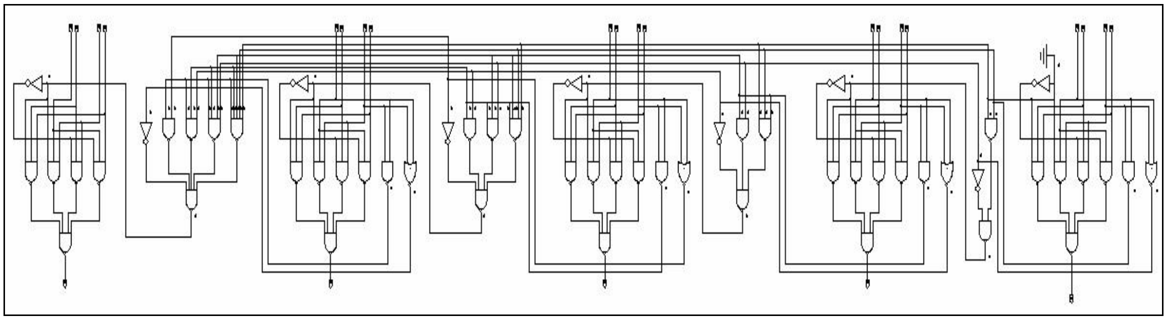


Figure 47. 5-Bit Adder Circuit Schematic in S-EDIT

2. Signals

The input and output signals of Figures 46 and 47 are:

- A0 through A4 and B0 through B4 represent binary numbers to be added together.

- ~ A0 through ~A4 and ~B0 through ~B4 represent the complement of the input signals supplied by the pipeline register preceding the adder.

- S0 through S4: Resulting binary number

3. Testing

Addition results in a 5-bit number, which ignores carry out for the final result. Table 9 shows the state table for the 5-Bit Adder in the decimal number system.

A	B	S
0	0 through 31	0 through 31
1	0 through 31	1 through 31, 0
2	0 through 31	2 through 31, 0,1
…	…	…
29	0 through 31	29, 30, 31,0,.., 28
30	0 through 31	30, 31,0,.., 29
31	0 through 31	31, 0,.., 30

Table 9. State Table for 5-Bit Adder

By using every possible value for A(4:0) and B(4:0) and by observing the output S (4:0), an exhaustive test was conducted. The graphical representation generated in Active HDL^{TM} is given in Figure 48. A part of the waveform generated is shown in Figure 49.

4. Verification

For a complete verification of the adder, an algorithm similar to the one used for the 12-Bit Comparator was applied. It is given in Table 10.

The Matlab results are compared to the simulation values and the operation of the 5-bit adder confirmed for every possible value by comparing the List File with a separate Matlab generated file.

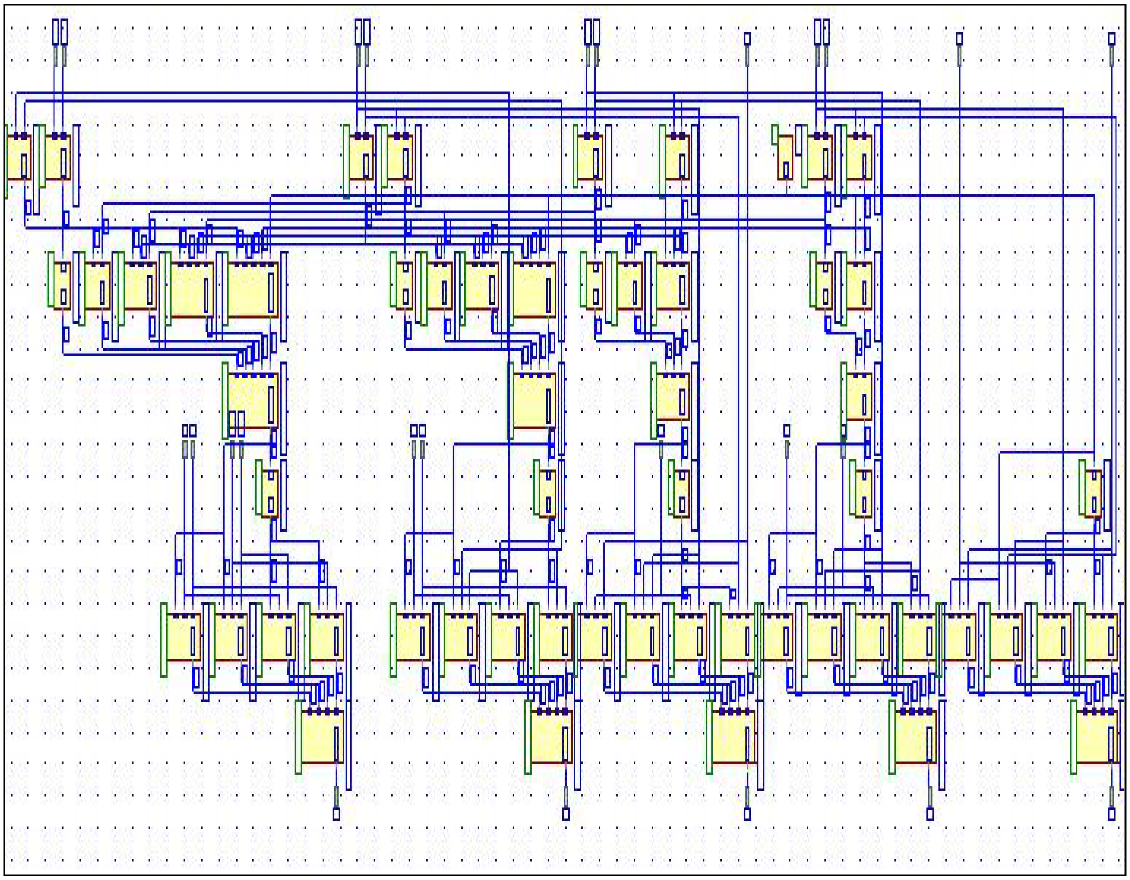
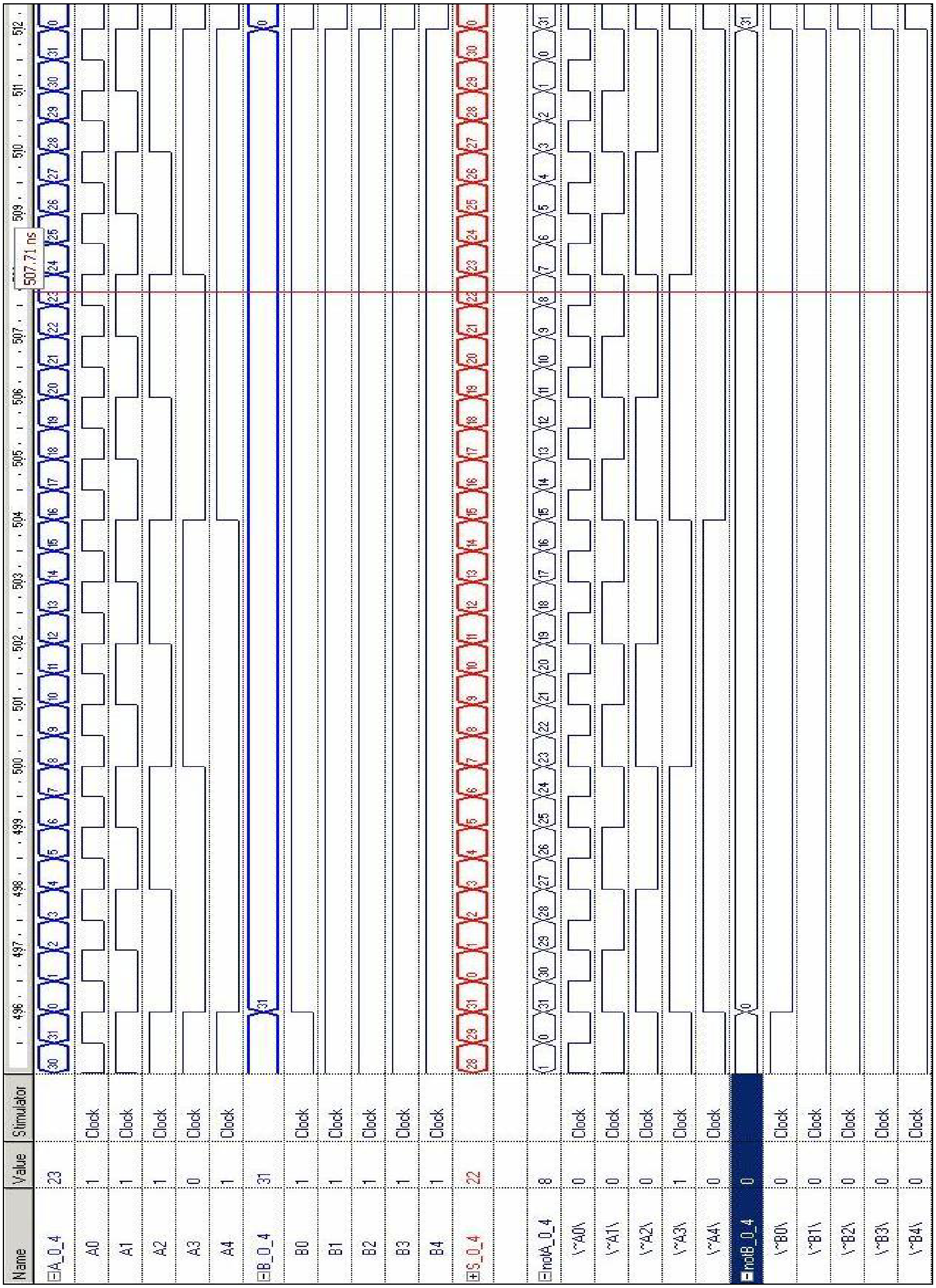


Figure 48.   5-Bit Adder Graphical Representation



Name	Value	Stimulator
A_0_4	23	
A0	1	Clock
A1	1	Clock
A2	1	Clock
A3	0	Clock
A4	1	Clock
B_0_4	31	
B0	1	Clock
B1	1	Clock
B2	1	Clock
B3	1	Clock
B4	1	Clock
S_0_4	22	
notA_0_4	8	
\~A0\	0	Clock
\~A1\	0	Clock
\~A2\	0	Clock
\~A3\	1	Clock
\~A4\	0	Clock
notB_0_4	0	
\~B0\	0	Clock
\~B1\	0	Clock
\~B2\	0	Clock
\~B3\	0	Clock
\~B4\	0	Clock

Figure 49. Waveform Showing Proper Operation for 5-Bit Adder

MATLAB	ACTIVE HDL^{TM} Waveform Editor Signal Stimulator Clock Periods			

for every_value of vector_A
begin
for every_value of Vector B
begin
Sum = vector_A + vector_B
if Sum>=2^5 then
Sum=Sum-2^5;
end;
end;
	A0	**2^0**	B0	**2^5**

	A1	**2^1**	B1	**2^6**
	A2	**2^2**	B2	**2^7**
	A3	**2^3**	B3	**2^8**
	A4	**2^4**	B4	**2^9**
	~A / ~B signals are generated with the complements of the clock signals above.			

Table 10. Exhaustive Test and Verification Algorithm for 5-Bit Adder

E. VERIFICATION OF 1 BIT 4-TO-1 MULTIPLEXER

1. Logic Symbol and Schematic

The phase samples for the cascade of RBP s can be sourced from four different sources. A 6-bit 4-to-1 multiplexer steers one of the inputs set into the RBP cascade. It consists of six identical 1-bit 4-to-1 multiplexers. The logic symbol and circuit schematic for one bit is shown in Figures 50 and 51, respectively.

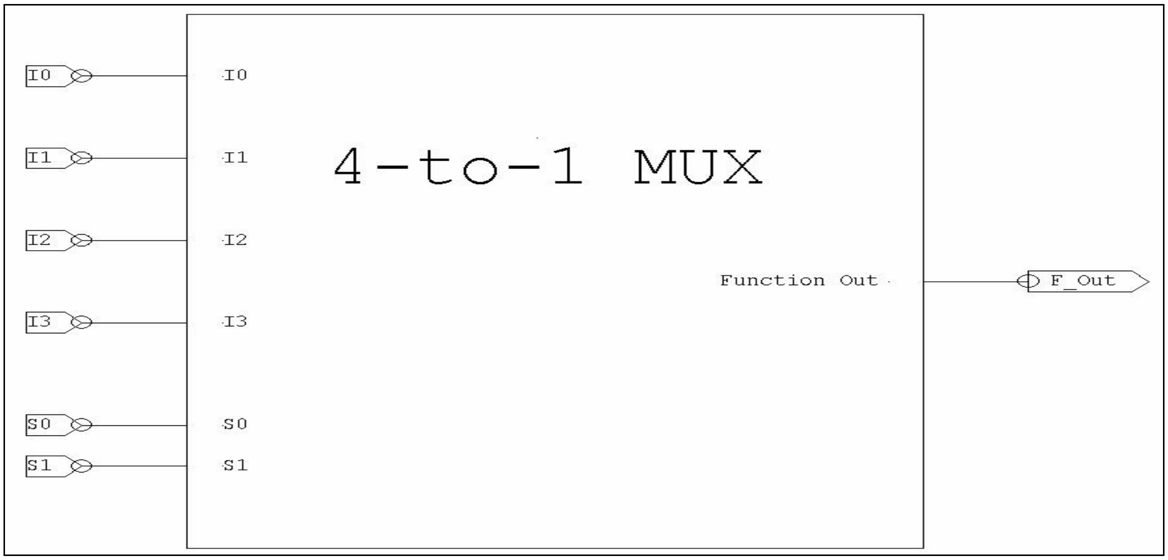


Figure 50. 1-Bit 4-to-1 Multiplexer Logic Symbol in S-EDIT

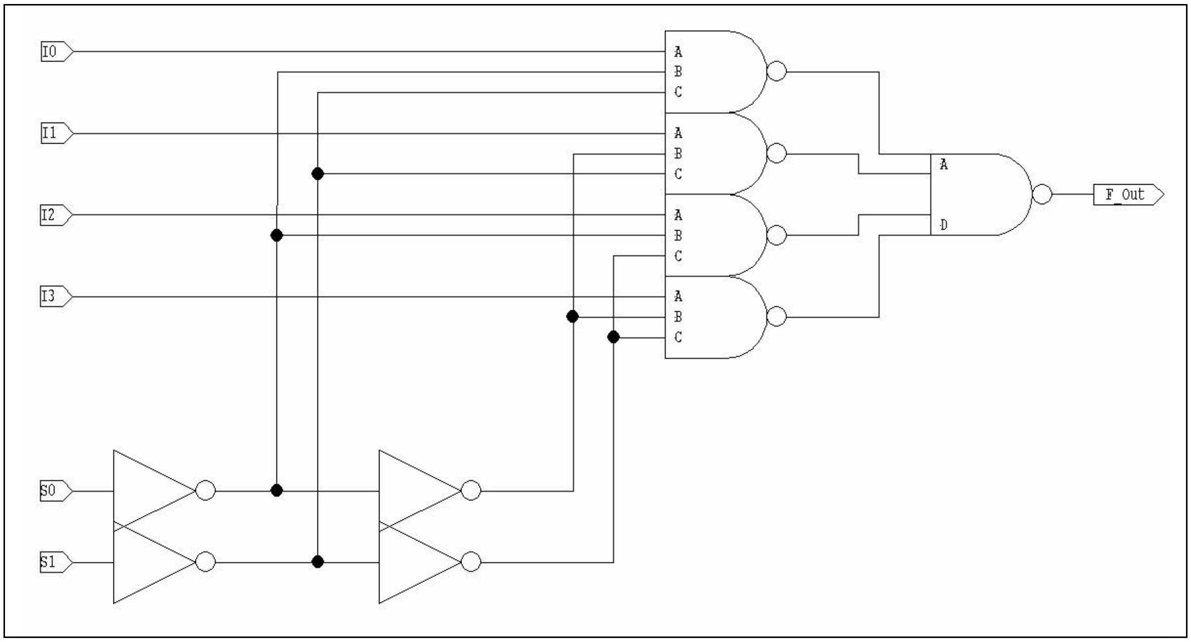


Figure 51. 1-Bit 4-to-1 Multiplexer Circuit Schematic in S-EDIT

2. Signals

The input and output signals of Figures 50 and 51 are:

- I0 through I3 are the signals into the multiplexer
- S1 and S0 are the signals that select the data to be steered
- F_Out is the selected signal among the inputs

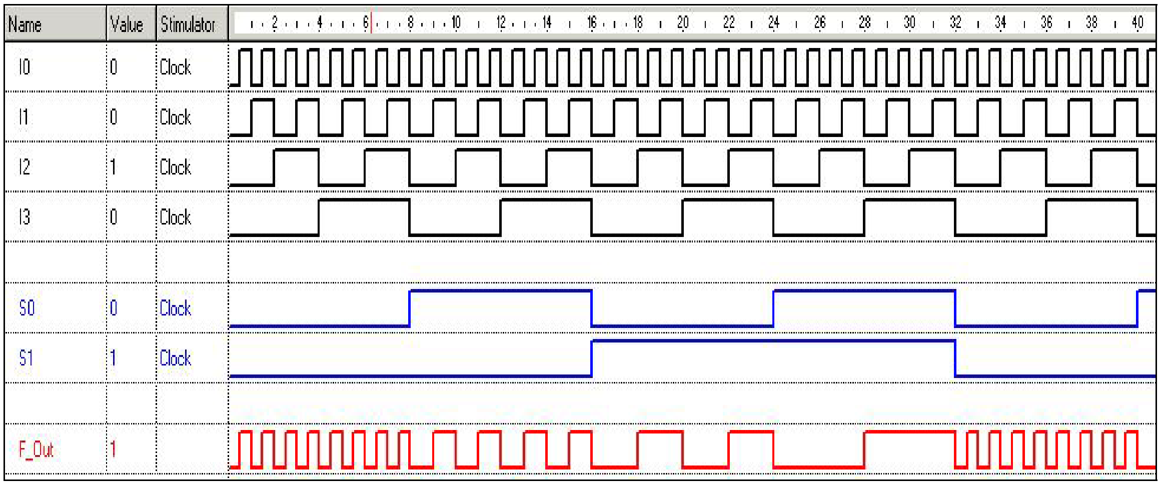
3. Testing

The state table for the multiplexer is given in Table 11. A complete testing was conducted using all possible input combinations.

I3	I2	I1	I0	S1	S0	F_OUT
D	C	B	A	0	0	A
D	C	B	A	0	1	B
D	C	B	A	1	0	C
D	C	B	A	1	1	D

Table 11. State Table for 1-Bit 4-to-1 Multiplexer

The waveform used is shown in Figure 52, while the graphical representation generated in Active HDL^{TM} is given in Figure 53.



Name	Value	Stimulator
I0	0	Clock
I1	0	Clock
I2	1	Clock
I3	0	Clock
S0	0	Clock
S1	1	Clock
F_Out	1	

Figure 52. Waveform Showing Proper Operation for the Multiplexer

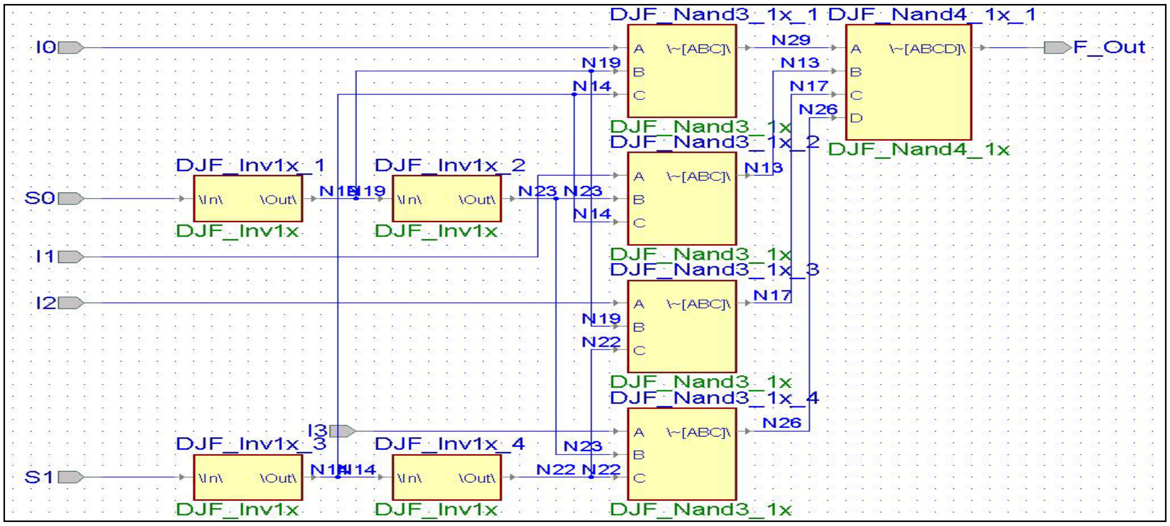


Figure 53. 1-Bit 4-to-1 Multiplexer Graphical Representation

4. Verification

In the waveform editor, one of the useful tools is the ability to compare waveforms. The procedure to compare two waveforms is as follows:

- Assign the desired input path number to S1 and S0 as a binary number. For instance, if it is desired to steer I0, force the control inputs to be (S1, S0) = (0,0) using a proper stimulator.

- Assign values for (I3:I0) in a counter fashion in the same way as in Figure 52 and run the simulation.

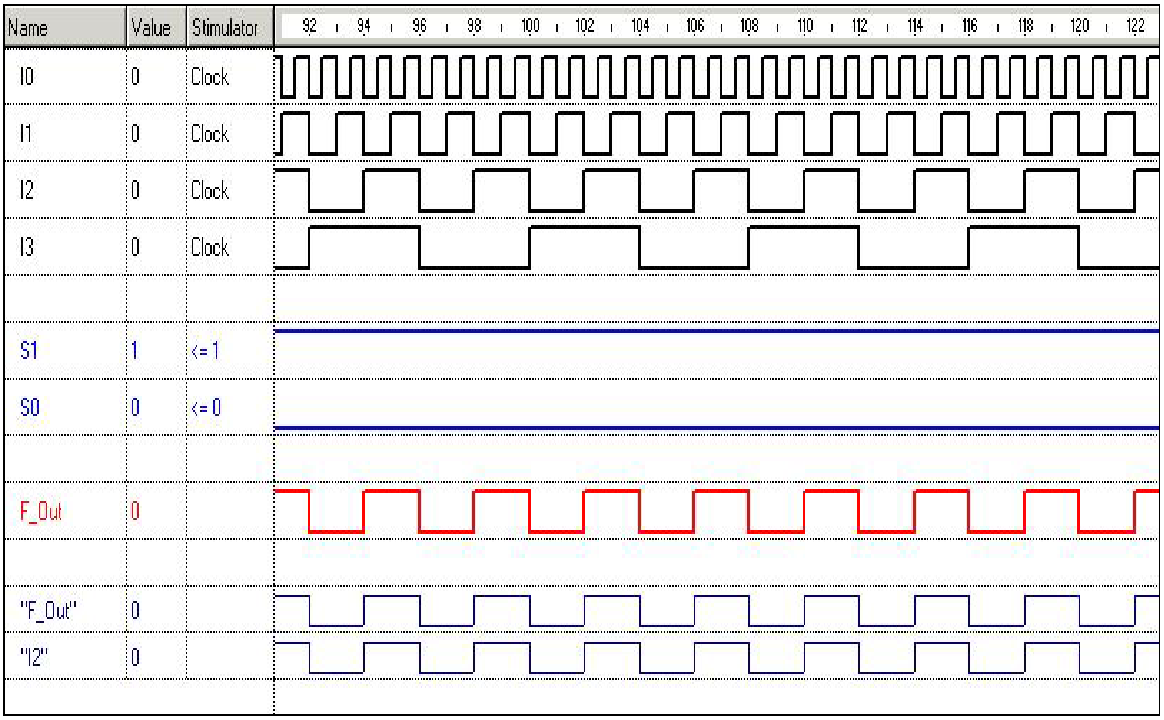
- Select F_Out and the proper I input at the same time by using the shift key and left mouse click.

- From the menu select Waveform > Compare Waveform

- There should not be any difference between the two waveforms.

- Repeat the procedure for all input paths using proper control signal values.

Using the procedure above, the circuit was tested and proved to be working properly. Figure 54 shows a part of the waveform used to test the I2 path by applying (1,0) for (S1, S0).



Name	Value	Stimulator
I0	0	Clock
I1	0	Clock
I2	0	Clock
I3	0	Clock
S1	1	<= 1
S0	0	<= 0
F_Out	0	
"F_Out"	0	
"I2"	0	

Figure 54. Exhaustive Test and Verification of 1-Bit 4-to-1 Multiplexer

This chapter has discussed the verification of the low level cells. VHDL Simulation results were compared against the C++ simulation results. The low level cells were tested and verified to operate properly. The next step in modeling and simulation of the DIS was the testing of the higher-level components and the data paths. Chapter V presents the methodology of testing the complete design.

THIS PAGE INTENTIONALLY LEFT BLANK

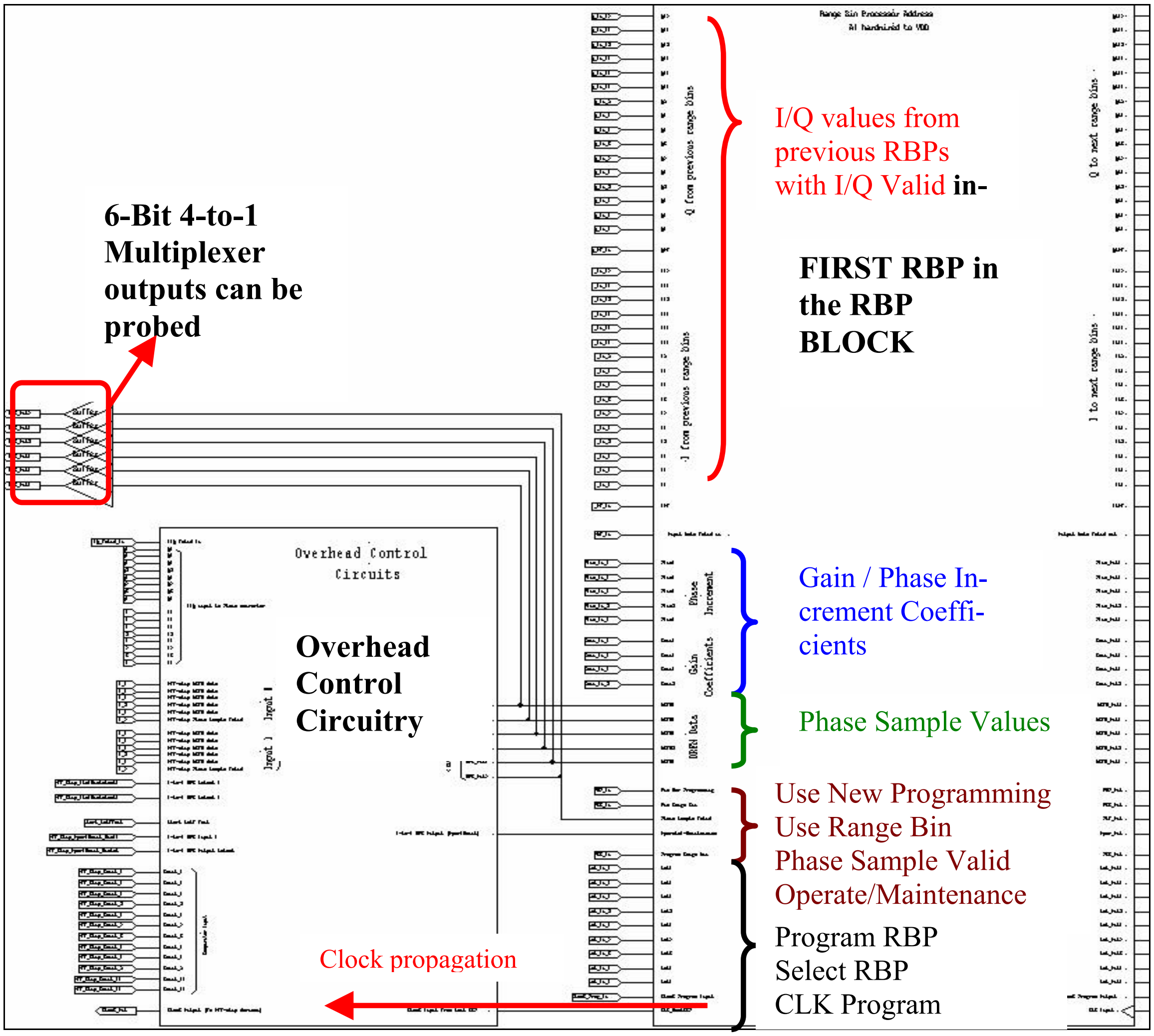
V. VHDL SIMULATIONS OF THE DIS CHIP

This chapter summarizes the tests conducted to verify the data paths and several functional blocks, self-test logic and phase extraction circuit. Overall DIS system was also tested with a 16 RBP block and verified to be functionally operating.

A. DATA FLOW PATHS

1. General View

Figure 55 contains the general block diagram of the Digital Image Synthesizer (DIS), with the overhead control circuitry.



6-Bit 4-to-1 Multiplexer outputs can be probed
I/Q values from previous RBPs with I/Q Valid in-
FIRST RBP in the RBP BLOCK
Overhead Control Circuits
Overhead Control Circuitry
Gain / Phase In-crement Coeffi-cients
Phase Sample Values
Use New Programming
Use Range Bin
Phase Sample Valid
Operate/Maintenance
Program RBP
Select RBP
CLK Program
Clock propagation

Figure 55. General Block Diagram of the DIS with the Overhead Control Circuitry

There are four data paths for the phase sample values. Two of them are for external input of the phase sample values into the Range Bin Processors (RBP s). Path 4 accomplishes the phase sample value extraction from in-phase (I) and quadrature (Q) inputs while Path 3 feeds the RBP s with test vectors to accomplish the self-test function. The inputs to those paths can be observed easily in the logic diagram of the overhead control circuitry in Figure 56, while the circuit schematic is given in Figure 57.

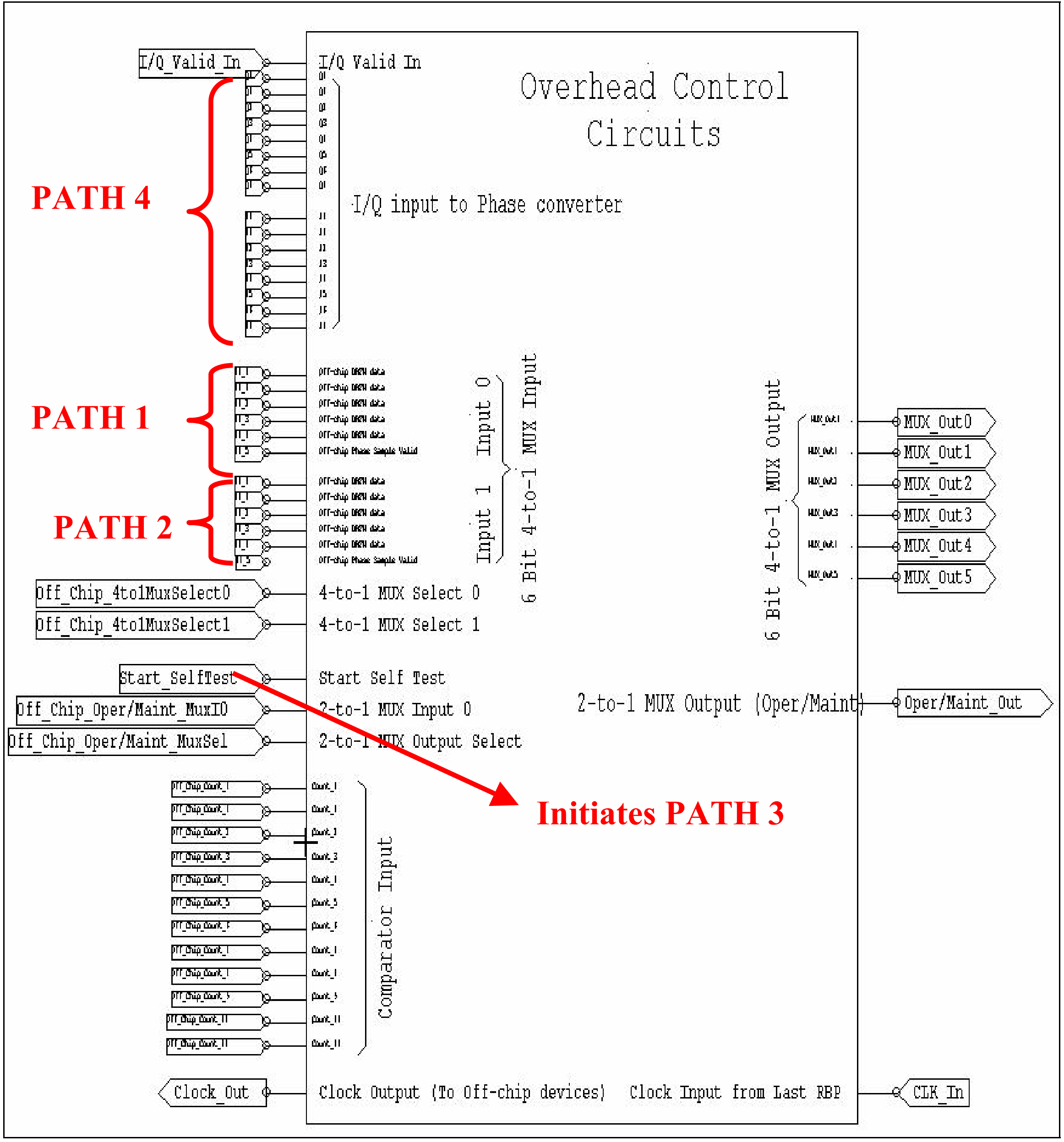


Figure 56. Logic Diagram of Overhead Control Circuitry

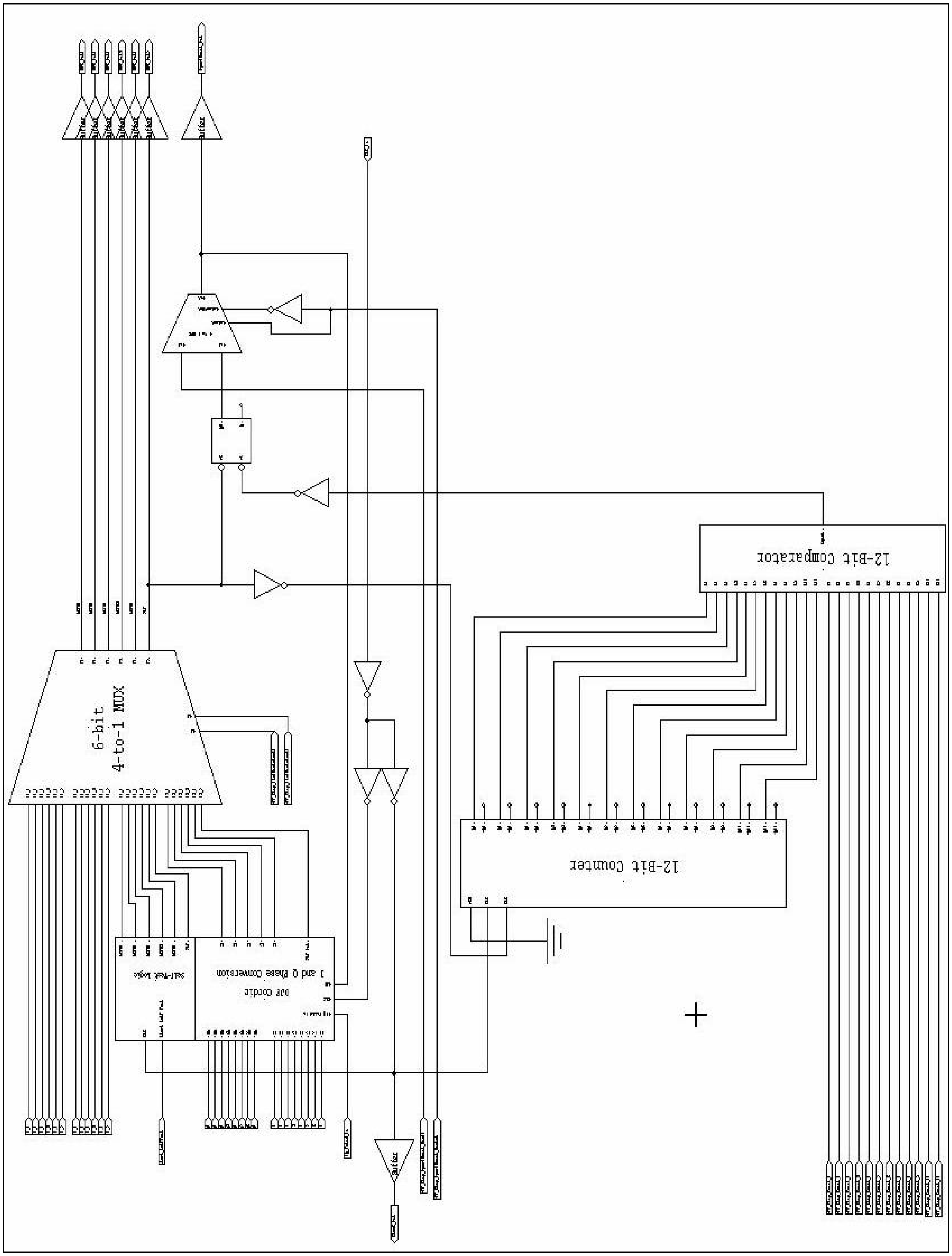


Figure 57. Circuit Schematic of Overhead Control Circuitry

2. Path 1 – External Phase Sample Values to RBPs

The first data path is shown in Figure 58. The off-chip inputs may be used directly to feed the Range Bin Processors (RBPs) with phase sample values. The 6-bit 4-to-1 multiplexer steers five-bit phase information along with the 1-bit Phase Sample Valid (PSV) signal that indicates a valid phase sample is ready to be processed. The 4-to-1 multiplexer select inputs should be programmed to choose the first path. The Operate/ Maintenance signal input to the RBP s is forced to "high" using a 1-bit 2-to-1 multiplexer select input and off-chip Operate/ Maintenance input. The former should be "low" while the latter is "high". Table 12 shows the control inputs and their required values to test the first path.

3. Path 2 – External Phase Sample Values to RBPs

The second data path is nothing but a duplicate of the Path 1. It may be used as a substitute for the first path and allows for future upgrades of the external circuitry. Path 2 can be seen in Figure 59 while Table 12 shows the control signals and their required values to test the path.

Control Inputs	PATH1	PATH2
	Required Value	Required Value
Off_Chip_4to1MuxSelect0	Low (0)	High (1)
Off_Chip_4to1MuxSelect1	Low (0)	Low (0)
Off_Chip_Oper/Maint_MuxIO	High (1)	High (1)
Off_Chip_Oper/Maint_MuxSel	Low (0)	Low (0)

Table 12. Control Signals to Test Path 1 and Path 2

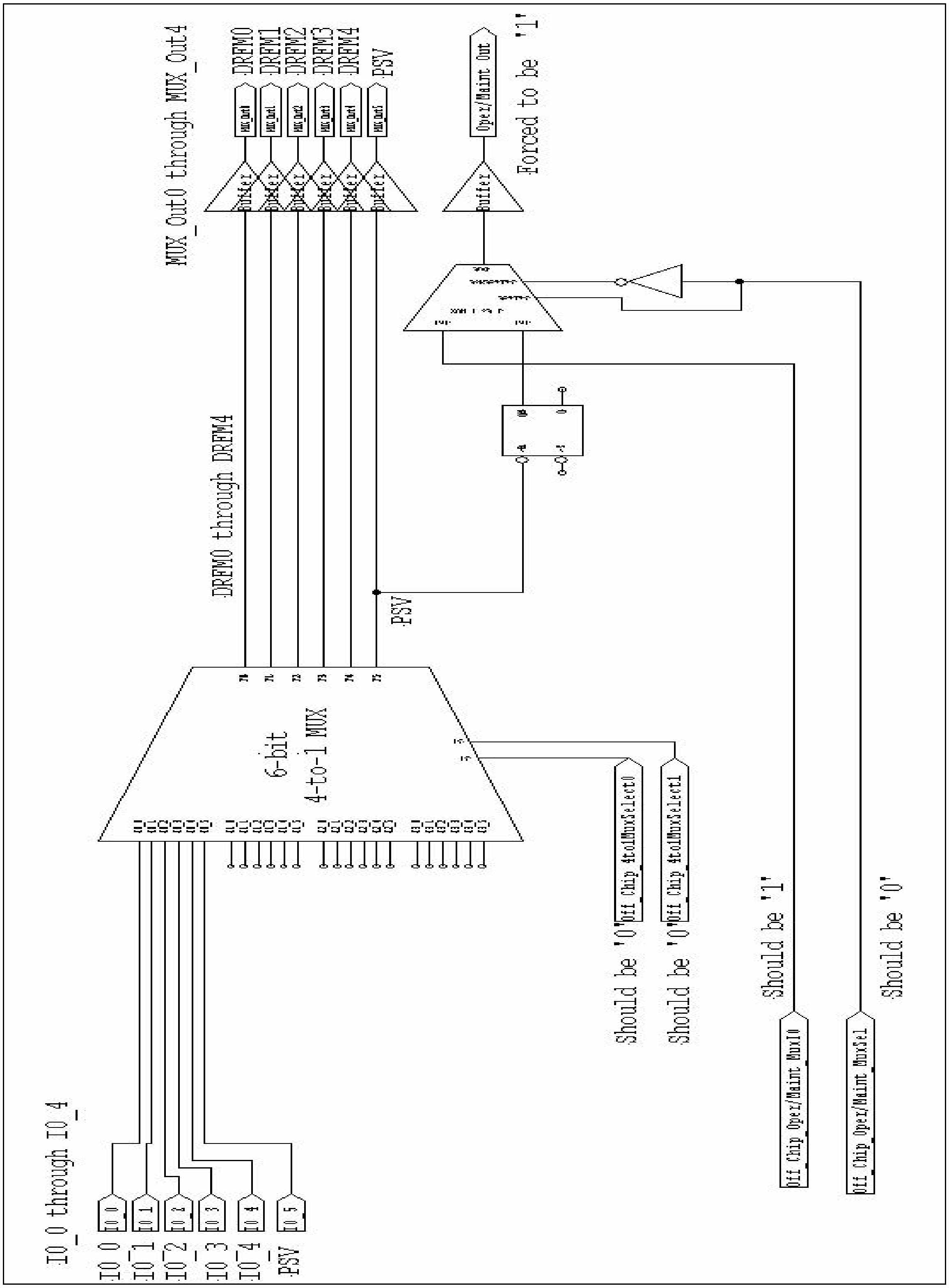


Figure 58. Data Path 1 – External Phase Sample Values

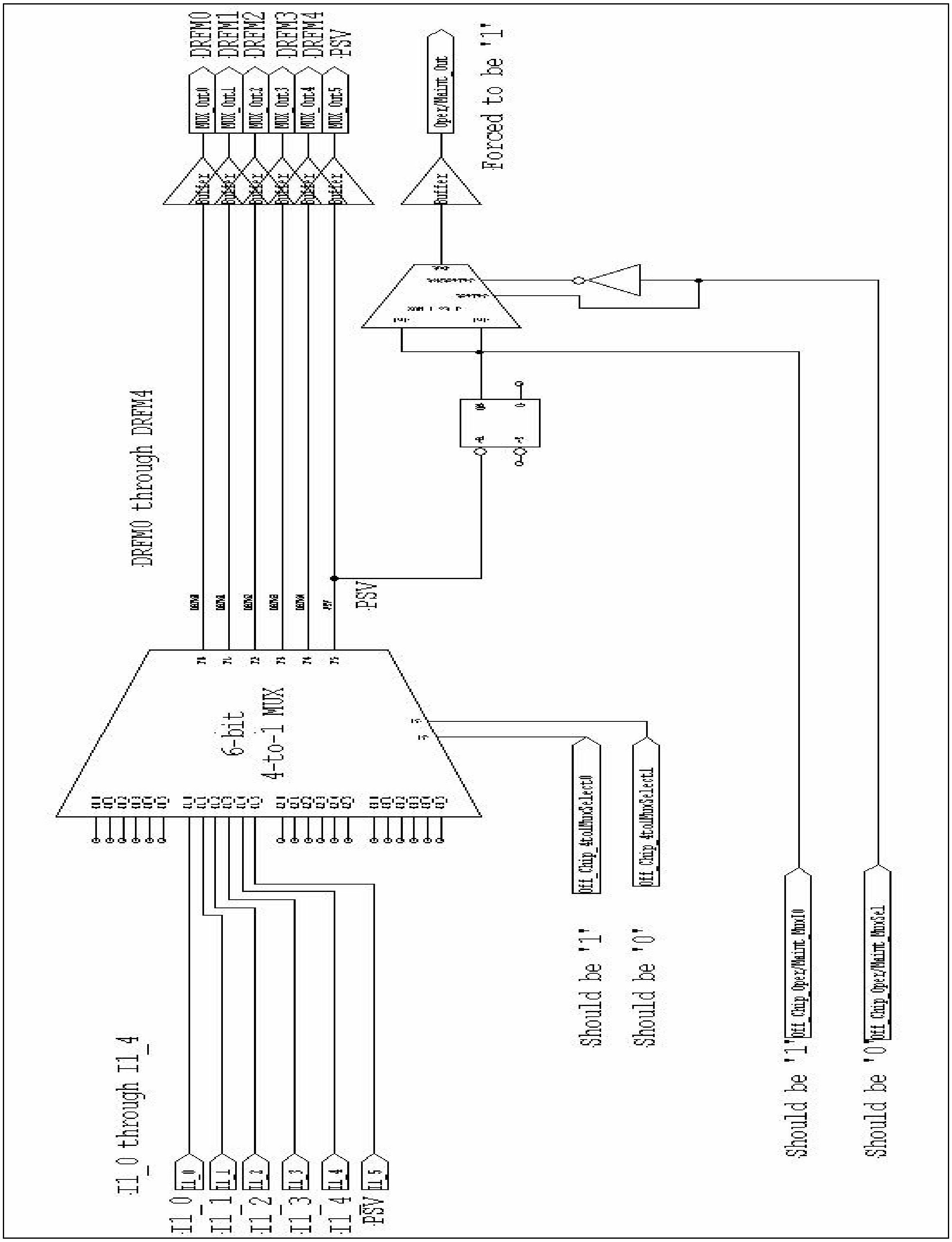


Figure 59. Data Path 2 – External Phase Sample Values

4. Path 3 – Phase Sample Values from Self Test Circuit to RBPs

The third path, shown in Figure 60, feeds the RBP s with automatically generated phase sample values as test vectors. Once the self-test sequence is started, it generates PSV and DRFM0 – DRFM4 outputs in a pseudo-random pattern. The proper target signature for the sequence is known by theory. By comparing the results of the self-test sequence and the ideal signature one can functionally test the DIS.

The Self-test mechanism is initiated by asserting the input Start_SelfTest. PSV output is "low" before the self-test starts, which causes the binary counter to be cleared. The number of the test vectors to be applied can be configured by the user; once self-test starts and PSV becomes "high" the binary counter starts to count upwards. Twelve off-chip inputs and the binary counter value are compared and when the values are equal, the Operate/Maintenance output becomes "low". I/Q values from the RBP s should "freeze" at the end of the self-test. However, since the test vectors from the Self-Test Logic are generated three clock cycles after the Start_SelfTest input goes "high", the off-chip number should be three greater than the desired test length. For instance, if the number of the self-test vectors to be generated is 61, the off-chip input should be 64.

The 2-to-1 multiplexer steers the Operate/Maintenance output to the RBP s via the ~S/~R latch. When the last test vector is generated, the comparator asserts the signal Equal, which sets the latch. The QN output of the latch becomes "low" and in turn, the Operate/Maintenance signal becomes "low". That freezes the target signature created in the RBP s.

As with the previous two paths, the control inputs to the 6-bit 4-to-1 multiplexer should be configured to select the self-test logic circuit outputs. The input Off_Chip_ Oper/MaintMuxIO can either be "low" or "high". The input Off_ChipOper/MaintMuxSel input selects the output of the latch for proper self-test operation. Table 13 shows the control signals and their values to test the path.

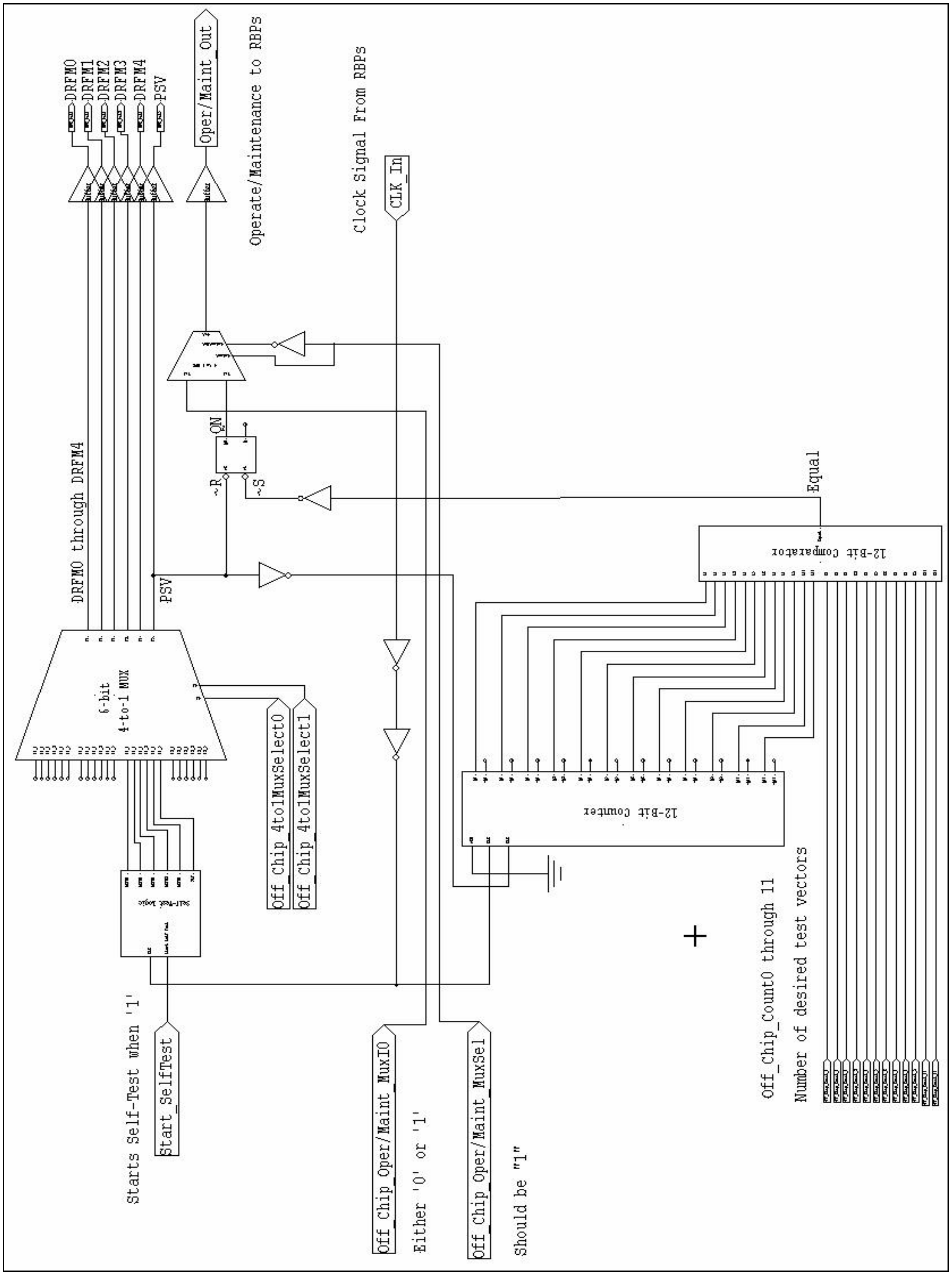


Figure 60. Data Path 3– Self-Test

Control Input	Required Value
Off_Chip_4to1MuxSelect0	Low (0)
Off_Chip_4to1MuxSelect1	High (1)
Off_Chip_Oper/Maint_MuxIO	Don't Care
Off_Chip_Oper/Maint_MuxSel	Low (0)
Off_Chip_Count0 through Off_Chip_Count11	Desired number of the test vectors
Start_SelfTest	Should be asserted to start the self-test sequence

Table 13. Control Signals to Test Path 3

5. Path 4 – Phase Sample Values from Phase Extraction Circuit to RBPs

The phase extraction circuit converts the I/Q values supplied by the Digital Radio Frequency Memory (DRFM) as eight-bit two's complement numbers into a corresponding phase angle value expressed as five-bit unsigned numbers for generating the false target signature.

The path from the phase extraction circuit to the RBP s is shown in Figure 61. The control inputs are given in Table 14. The extraction is enabled with the assertion of the signal I/Q_Valid_In. The DRFM values are loaded continuously since the Load input of the phase extraction circuit is hard-wired to "high".

The 6-bit 4-to-1 multiplexer should be controlled so that proper data is transferred to the RBP s. The 2-to-1 multiplexer passes a "high" for the Operate/Maintenance signal into the RBP s. The I/Q values are off-chip signals coming from the DRFM.

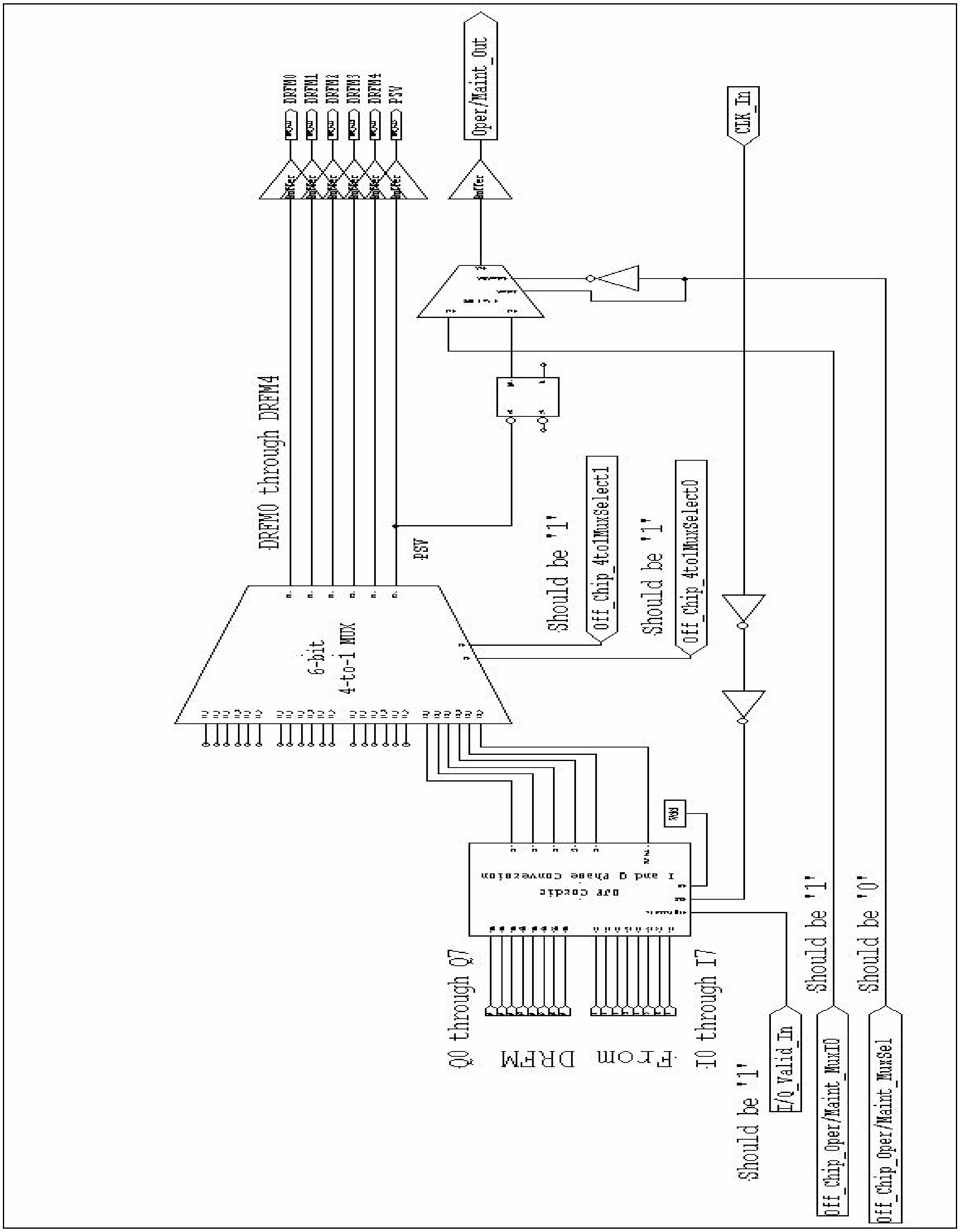


Figure 61. Data Path 4– Phase Extraction

Control Input	Required Value
Off_Chip_4to1MuxSelect0	High (1)
Off_Chip_4to1MuxSelect1	High (1)
Off_Chip_Oper/Maint_MuxIO	High (1)
Off_Chip_Oper/Maint_MuxSel	Low (0)
I/Q_Valid_In	Asserted to start phase extraction

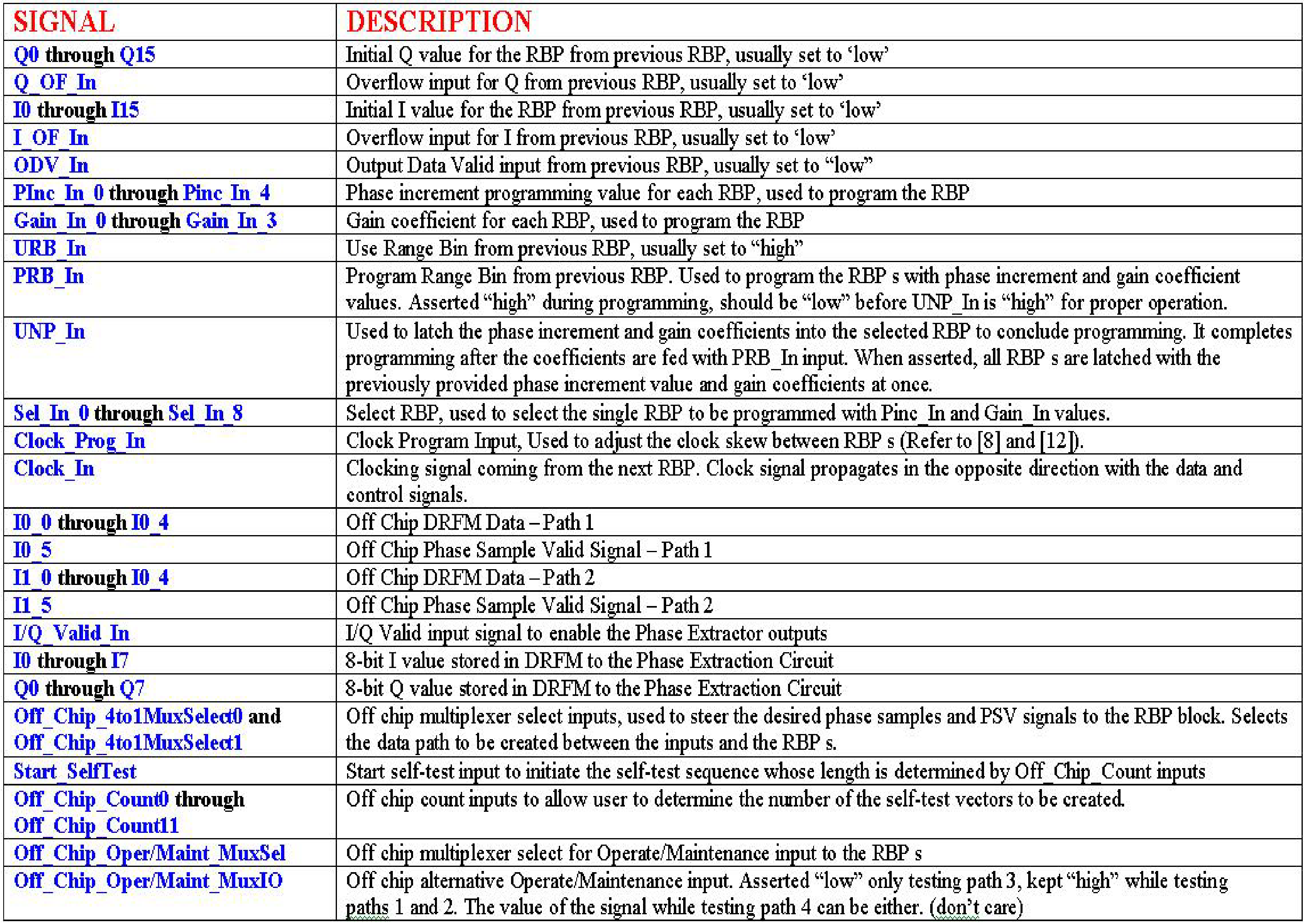
Table 14. Control Signals to Test Path 4

B. INPUT / OUTPUT SIGNALS

A number of input and output signals must be instantiated with specific values at certain times in order to drive the simulation. Table 15 lists the input signals and their functions while Table 16 shows the output signals and their descriptions.

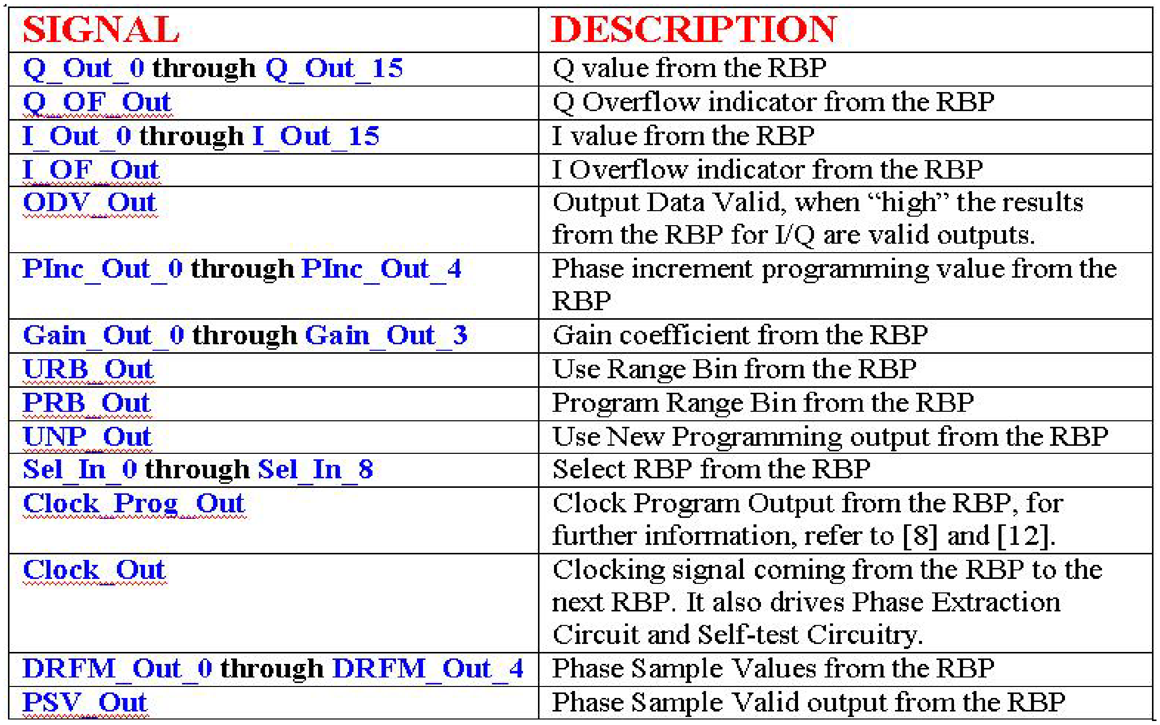
C. SIMULATIONS

The simulations performed on the Range Bin Processors include simulations on a single RBP and 4 and 16 cascaded RBP s. Due to the memory allocation problem on initialization in the simulator software used, Aldec Active HDL^{TM}, simulations with 256 RBP s and 512 RBP s could not been conducted. However, because all RBP s are identical in hardware design and programming style and input/output signal propagation, even four RBP s cascaded together can represent the circuit behavior of all serial 512 RBP s. Simulations involving the control circuitry and the different data paths are conducted with four RBP s cascaded together, representing the overall DIS. The important features of the DIS, Self-test Logic and Phase Extraction Circuit, are tested separately before they are integrated with the rest of the circuit.



SIGNAL	DESCRIPTION
Q0 through Q15	Initial Q value for the RBP from previous RBP, usually set to 'low'
Q_OF_In	Overflow input for Q from previous RBP, usually set to 'low'
I0 through I15	Initial I value for the RBP from previous RBP, usually set to 'low'
I_OF_In	Overflow input for I from previous RBP, usually set to 'low'
ODV_In	Output Data Valid input from previous RBP, usually set to "low"
PInc_In_0 through Pinc_In_4	Phase increment programming value for each RBP, used to program the RBP
Gain_In_0 through Gain_In_3	Gain coefficient for each RBP, used to program the RBP
URB_In	Use Range Bin from previous RBP, usually set to "high"
PRB_In	Program Range Bin from previous RBP. Used to program the RBP s with phase increment and gain coefficient values. Asserted "high" during programming, should be "low" before UNP_In is "high" for proper operation.
UNP_In	Used to latch the phase increment and gain coefficients into the selected RBP to conclude programming. It completes programming after the coefficients are fed with PRB_In input. When asserted, all RBP s are latched with the previously provided phase increment value and gain coefficients at once.
Sel_In_0 through Sel_In_8	Select RBP, used to select the single RBP to be programmed with Pinc_In and Gain_In values.
Clock_Prog_In	Clock Program Input, Used to adjust the clock skew between RBP s (Refer to [8] and [12]).
Clock_In	Clocking signal coming from the next RBP. Clock signal propagates in the opposite direction with the data and control signals.
I0_0 through I0_4	Off Chip DRFM Data – Path 1
I0_5	Off Chip Phase Sample Valid Signal – Path 1
I1_0 through I0_4	Off Chip DRFM Data – Path 2
I1_5	Off Chip Phase Sample Valid Signal – Path 2
I/Q_Valid_In	I/Q Valid input signal to enable the Phase Extractor outputs
I0 through I7	8-bit I value stored in DRFM to the Phase Extraction Circuit
Q0 through Q7	8-bit Q value stored in DRFM to the Phase Extraction Circuit
Off_Chip_4to1MuxSelect0 and Off_Chip_4to1MuxSelect1	Off chip multiplexer select inputs, used to steer the desired phase samples and PSV signals to the RBP block. Selects the data path to be created between the inputs and the RBP s.
Start_SelfTest	Start self-test input to initiate the self-test sequence whose length is determined by Off_Chip_Count inputs
Off_Chip_Count0 through Off_Chip_Count11	Off chip count inputs to allow user to determine the number of the self-test vectors to be created.
Off_Chip_Oper/Maint_MuxSel	Off chip multiplexer select for Operate/Maintenance input to the RBP s
Off_Chip_Oper/Maint_MuxIO	Off chip alternative Operate/Maintenance input. Asserted "low" only testing path 3, kept "high" while testing paths 1 and 2. The value of the signal while testing path 4 can be either. (don't care)

Table 15. Input Signals to the Digital Image Synthesizer



SIGNAL	DESCRIPTION
Q_Out_0 through Q_Out_15	Q value from the RBP
Q_OF_Out	Q Overflow indicator from the RBP
I_Out_0 through I_Out_15	I value from the RBP
I_OF_Out	I Overflow indicator from the RBP
ODV_Out	Output Data Valid, when "high" the results from the RBP for I/Q are valid outputs.
PInc_Out_0 through PInc_Out_4	Phase increment programming value from the RBP
Gain_Out_0 through Gain_Out_3	Gain coefficient from the RBP
URB_Out	Use Range Bin from the RBP
PRB_Out	Program Range Bin from the RBP
UNP_Out	Use New Programming output from the RBP
Sel_In_0 through Sel_In_8	Select RBP from the RBP
Clock_Prog_Out	Clock Program Output from the RBP, for further information, refer to [8] and [12].
Clock_Out	Clocking signal coming from the RBP to the next RBP. It also drives Phase Extraction Circuit and Self-test Circuitry.
DRFM_Out_0 through DRFM_Out_4	Phase Sample Values from the RBP
PSV_Out	Phase Sample Valid output from the RBP

Table 16. Output Signals from the Digital Image Synthesizer

1. Simulation of a Single RBP

A single RBP, shown in Figure 62, is functionally tested and verified. The simulation results are compared with the ideal outputs that are computed using C++ by Prof. Fouts. For details on the design of the RBP s, refer to [12].

The pipeline registers inside the RBP s should be cleared prior to introducing valid phase sample values to the RBP. In order to accomplish this task, the circuit should be clocked *N* times where *N* is the sum of the number of pipeline stages and the number of the RBP s.

Moreover, the delay signal should be initialized. The signal "Delay" is the input Clk_Prog_In stored in a bit in the 6-bit register. Because there is a feedback to the Clock Splitting circuit from the 6-bit register, the VHDL programmer should initialize the value for the Delay signal to either '0' or '1' to create a valid clock signal to the RBP. The "Delay" signal is also shown in Figure 62.

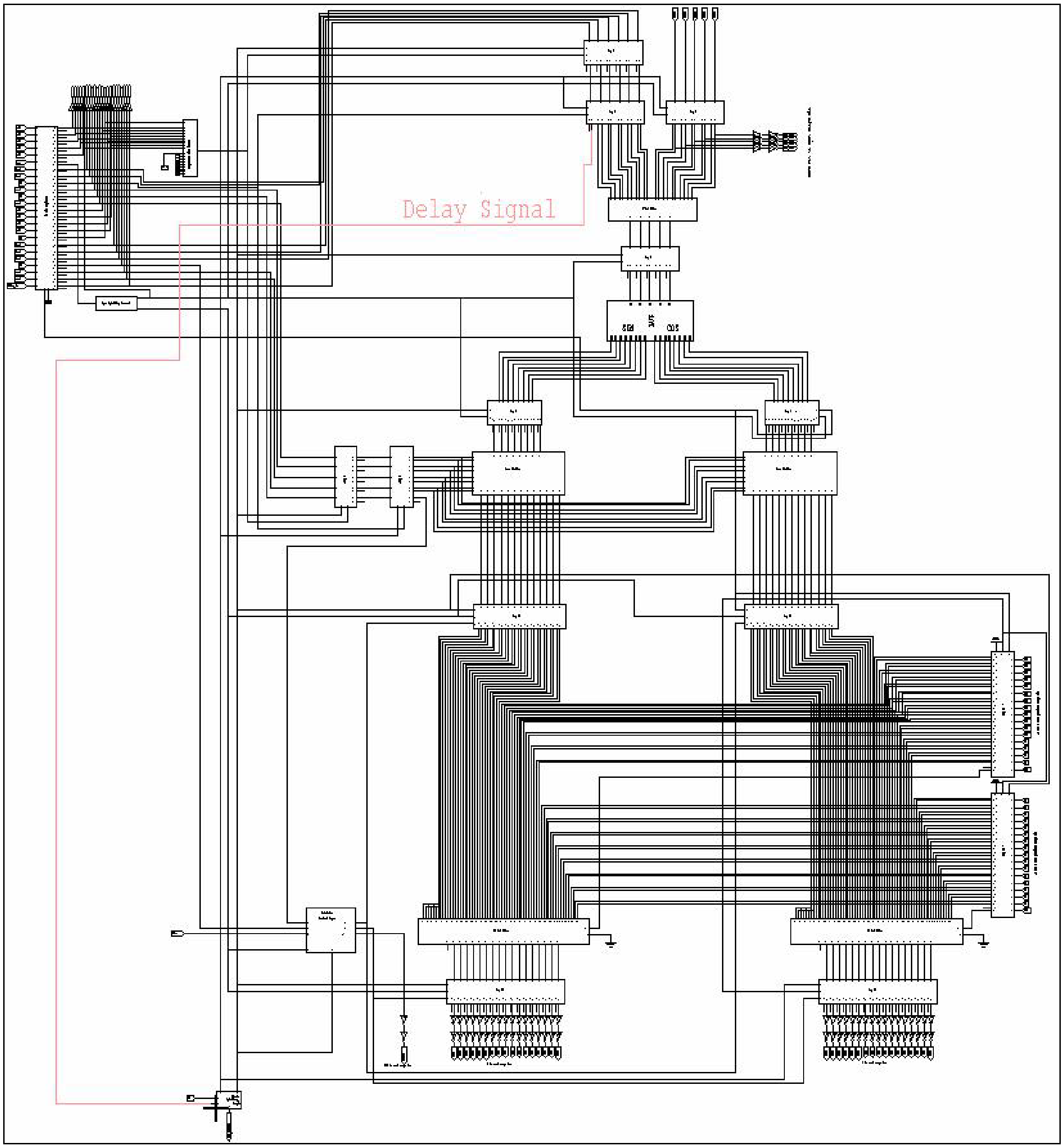


Figure 62. A Single Range Bin Processor Schematic and Delay Signal in S-Edit

The simulation algorithm is as follows:

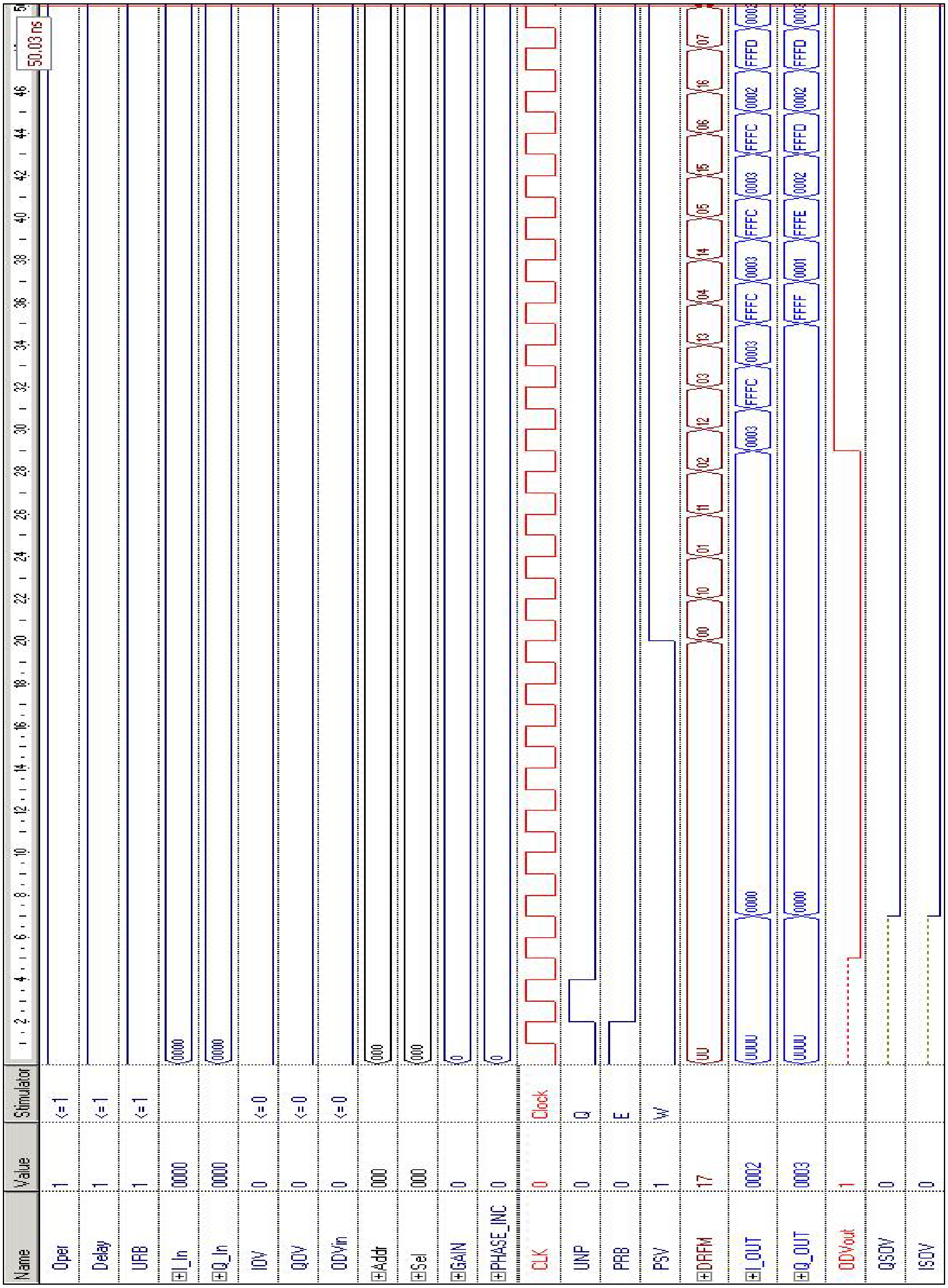
- Set Addr0 through Addr8 = ‘0’, the address of the RBP. In the actual hardware implementation, this is accomplished by hardwiring address lines to Vdd or Gnd. The VHDL programmer need not assign any values.

- Set Clock Rate, CLK = Stimulator→ Clock→ 2ns.

- Set Clk_Prg_In, I0 through I15, Q0 through Q15, IOV, QOV, ODVIn and Sel0 through Sel8 =’0’ by Stimulator→Value→0
- Set Delay=’1’. (Delay signal is in the *entity* DTM_ClockSplitter_1)
- Set Oper, URB = ‘1’ by Stimulator → Value →1
- Set PSV=0, UNP=0, PRB=0
- Clock RPB for 5 times to clear the pipeline
- Set PRB = ‘1’ and Gain0 through Gain3=’0’, PInc0 through PInc4=’0’
- Clock RBP once
- Set PRB = ‘0’, UNP=’1’
- Clock RBP once
- Set UNP = ‘0’
- Clock RBP until ISOV, QSOV and ODVOut are ‘0’
- Set PSV =’1’ and DRFM0 through DRFM4 to the desired phase sample values, clock RBP, repeat for every DRFM sample value
- Set PSV= ‘0’
- Clock RBP until ODVOut = ‘0’
- Watch and record the values for IS0 through IS15, QS0 through QS15, ISOV (Overflow), QSOV (Overflow) and ODV_Out
- Compare the results with the C++ outputs
- If they don’t match, use the Active HDL^{TM} Block Diagram Editor to trace the signals and find the problem. If they match, document the results.

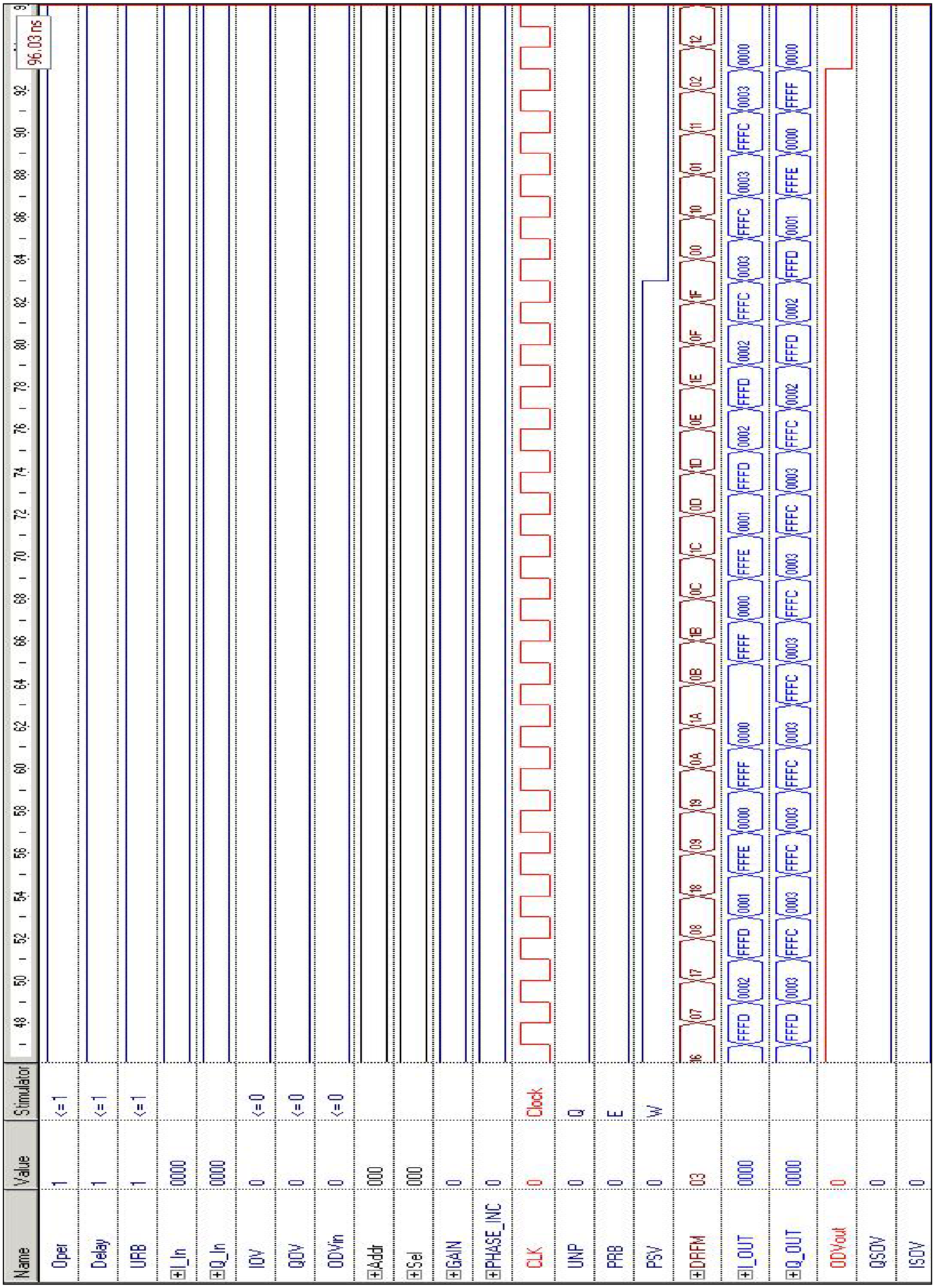
The waveform used in the simulation is shown in Figure 63.

Table 17 presents the programming coefficients for the RBP, the simulation results and the comparison with the C++ outputs.



Name	Value	Stimulator
Oper	1	<=1
Delay	1	<=1
URB	1	<=1
⊞I_In	0000	
⊞Q_In	0000	
IOV	0	<=0
QOV	0	<=0
ODVin	0	<=0
⊞Addr	000	
⊞Sel	000	
⊞GAIN	0	
⊞PHASE_INC	0	
CLK	0	Clock
UNP	0	Q
PRB	0	E
PSV	1	W
⊞DRFM	17	
⊞I_OUT	0002	
⊞Q_OUT	0003	
ODVout	1	
QSOV	0	
ISOV	0	

Figure 63. Simulation of a Single RBP



Name	Value	Stimulator
Oper	1	<=1
Delay	1	<=1
URB	1	<=1
⊞I_In	0000	
⊞Q_In	0000	
IOV	0	<=0
QOV	0	<=0
ODVin	0	<=0
⊞Addr	000	
⊞Sel	000	
⊞GAIN	0	
⊞PHASE_INC	0	
CLK	0	Clock
UNP	0	Q
PRB	0	E
PSV	0	W
⊞DRFM	03	
⊞I_OUT	0000	
⊞Q_OUT	0000	
ODVout	0	
QSOV	0	
ISOV	0	

48 50 52 54 56 58 60 62 64 66 68 70 72 74 76 78 80 82 84 86 88 90 92 9
96.03 ns
06 07 17 08 18 09 19 0A 1A 0B 1B 0C 1C 0D 1D 0E 1E 0F 1F 00 10 01 11 02 12
FFFD 0002 FFFD 0001 FFFE 0000 FFFF 0000 FFFF 0000 FFFE 0001 FFFD 0002 FFFD 0002 FFFC 0003 FFFC 0003 FFFC 0003 0000
FFFD 0003 FFFC 0003 FFFC 0003 FFFC 0003 FFFC 0003 FFFC 0003 FFFC 0003 FFFC 0002 FFFD 0002 FFFD 0001 FFFE 0000 FFFF 0000

Simulation of a Single RBP, Continued

1 RBP	Simulation Results				C++ Outputs			
Gain = 0 PInc = 0								
Phase Samples (Hex)	I_Out (Hex)	Q_Out (Hex)	I_OF _Out	Q_OF _Out	I_Out (Hex)	Q_Out (Hex)	I_OF _Out	Q_OF _Out
00	0003	0000	0	0	0003	0000	0	0
10	FFFC	0000	0	0	FFFC	0000	0	0
01	0003	0000	0	0	0003	0000	0	0
11	FFFC	FFFF	0	0	FFFC	FFFF	0	0
02	0003	0001	0	0	0003	0001	0	0
12	FFFC	FFFE	0	0	FFFC	FFFE	0	0
03	0003	0002	0	0	0003	0002	0	0
13	FFFC	FFFD	0	0	FFFC	FFFD	0	0
04	0002	0002	0	0	0002	0002	0	0
14	FFFD	FFFD	0	0	FFFD	FFFD	0	0
05	0002	0003	0	0	0002	0003	0	0
15	FFFD	FFFC	0	0	FFFD	FFFC	0	0
06	0001	0003	0	0	0001	0003	0	0
16	FFFE	FFFC	0	0	FFFE	FFFC	0	0
07	0000	0003	0	0	0000	0003	0	0
17	FFFF	FFFC	0	0	FFFF	FFFC	0	0
08	0000	0003	0	0	0000	0003	0	0
18	0000	FFFC	0	0	0000	FFFC	0	0
09	FFFF	0003	0	0	FFFF	0003	0	0
19	0000	FFFC	0	0	0000	FFFC	0	0
0A	FFFE	0003	0	0	FFFE	0003	0	0
1A	0001	FFFC	0	0	0001	FFFC	0	0
0B	FFFD	0003	0	0	FFFD	0003	0	0
1B	0002	FFFC	0	0	0002	FFFC	0	0
0C	FFFD	0002	0	0	FFFD	0002	0	0
1C	0002	FFFD	0	0	0002	FFFD	0	0
0D	FFFC	0002	0	0	FFFC	0002	0	0
1D	0003	FFFD	0	0	0003	FFFD	0	0
0E	FFFC	0001	0	0	FFFC	0001	0	0
1E	0003	FFFE	0	0	0003	FFFE	0	0
0F	FFFC	0000	0	0	FFFC	0000	0	0
1F	0003	FFFF	0	0	0003	FFFF	0	0

Table 17. Simulation Results and Comparison for a Single RBP

2. Simulation of 4 RBP s in Series

The simulation algorithm is as follows:

- Set Clock Rate, Clock_In = Stimulator→ Clock → 2ns

- Set Clk_Prg_In, I_In0 through I_In15, Q0 through Q_In15, I_OF_In, Q_OF_In, ODV_In by Stimulator→Value→0

- Set Delay='0' in all clock splitting circuits in every RBP using the design browser and by adding signal names into the waveform editor. (Delay signals are in the *entities* DJF_ClockTrue_1 and DJF_ClockComp_1)

- Set Oper_In, URB_In = '1' by Stimulator → Value →1

- Set PSV_In=0, UNP_In=0, PRB_In=0

- Clock RPB for 11 times to clear the pipeline

- Set PRB_In = '1'

- Select the RBP to be programmed using the Sel_In0 through Sel_In8 inputs. Set Gain_In0 through Gain_In3='0',and PInc_In0 through PInc_In4='0' to the desired values for the RBP to be programmed. Clock the RBP once. Repeat for all RBP s.

- Set PRB_In = '0', UNP_In='1'

- Clock RBP once

- Set UNP_In = '0'

- Clock RBP until I_OF_Out, Q_OF_Out and ODV_Out are '0'

- Set PSV_In ='1' and DRFM_In0 through DRFM_In4 to the desired phase sample values, clock RBP, repeat for every DRFM sample value

- Set PSV_In= '0', clock RBP until ODV_Out = '0'

- Watch and record the values for I_Out0 through I_Out15, Q_Out0 through Q_Out15, I_OF_Out (Overflow), Q_OF_Out (Overflow) and ODV_Out

- Compare the results with the C++ outputs.

The waveform used in the simulation for 4 RBP s in series is shown in Figure 64. Table 18 shows the programming coefficients for the RBP s, the simulation results, and the comparison with the C++ outputs.

RBP	0		1		2		3	
Gain\(Hex)	00		04		08		0C	
PInc(Hex)	00		08		10		18	
	Simulation Results				C++ Outputs			
Phase Samples (Hex)	I_Out (Hex)	Q_Out (Hex)	I_OF _Out	Q_OF _Out	I_Out (Hex)	Q_Out (Hex)	I_OF _Out	Q_OF _Out
00	0003	0000	0	0	0003	0000	0	0
10	FFFC	001F	0	0	FFFC	001F	0	0
01	FFC3	FFE0	0	0	FFC3	FFE0	0	0
11	0034	FE22	0	0	0034	FE22	0	0
02	FFCA	01D0	0	0	FFCA	01D0	0	0
12	0091	FE33	0	0	0091	FE33	0	0
03	FF70	01BF	0	0	FF70	01BF	0	0
13	00E8	FE5B	0	0	00E8	FE5B	0	0
04	FF1A	0197	0	0	FF1A	0197	0	0
14	0137	FE8E	0	0	0137	FE8E	0	0
05	FECF	0167	0	0	FECF	0167	0	0
15	0177	FED2	0	0	0177	FED2	0	0
06	FE8F	0124	0	0	FE8F	0124	0	0
16	01AB	FF21	0	0	01AB	FF21	0	0
07	FE5C	00D7	0	0	FE5C	00D7	0	0
17	01CB	FF78	0	0	01CB	FF78	0	0
08	FE3E	0081	0	0	FE3E	0081	0	0
18	01E0	FFD6	0	0	01E0	FFD6	0	0
09	FE2A	0027	0	0	FE2A	0027	0	0
19	01DC	0034	0	0	01DC	0034	0	0
0A	FE2D	FFCA	0	0	FE2D	FFCA	0	0
1A	01CA	0091	0	0	01CA	0091	0	0
0B	FE3E	FF70	0	0	FE3E	FF70	0	0
1B	01A2	00E8	0	0	01A2	00E8	0	0
0C	FE66	FF1A	0	0	FE66	FF1A	0	0
1C	016F	0137	0	0	016F	0137	0	0
0D	FE97	FECF	0	0	FE97	FECF	0	0
1D	012C	0177	0	0	012C	0177	0	0
0E	FEDA	FE8F	0	0	FEDA	FE8F	0	0
1E	00DD	01AB	0	0	00DD	01AB	0	0
0F	FF26	FE5C	0	0	FF26	FE5C	0	0
1F	0085	01CB	0	0	0085	01CB	0	0
-	FF80	FE3E	0	0	FF80	FE3E	0	0
-	0025	0200	0	0	0025	0200	0	0
-	FF9C	FE0C	0	0	FF9C	FE0C	0	0

Table 18. Simulation Results and Comparison for 4 RBP s

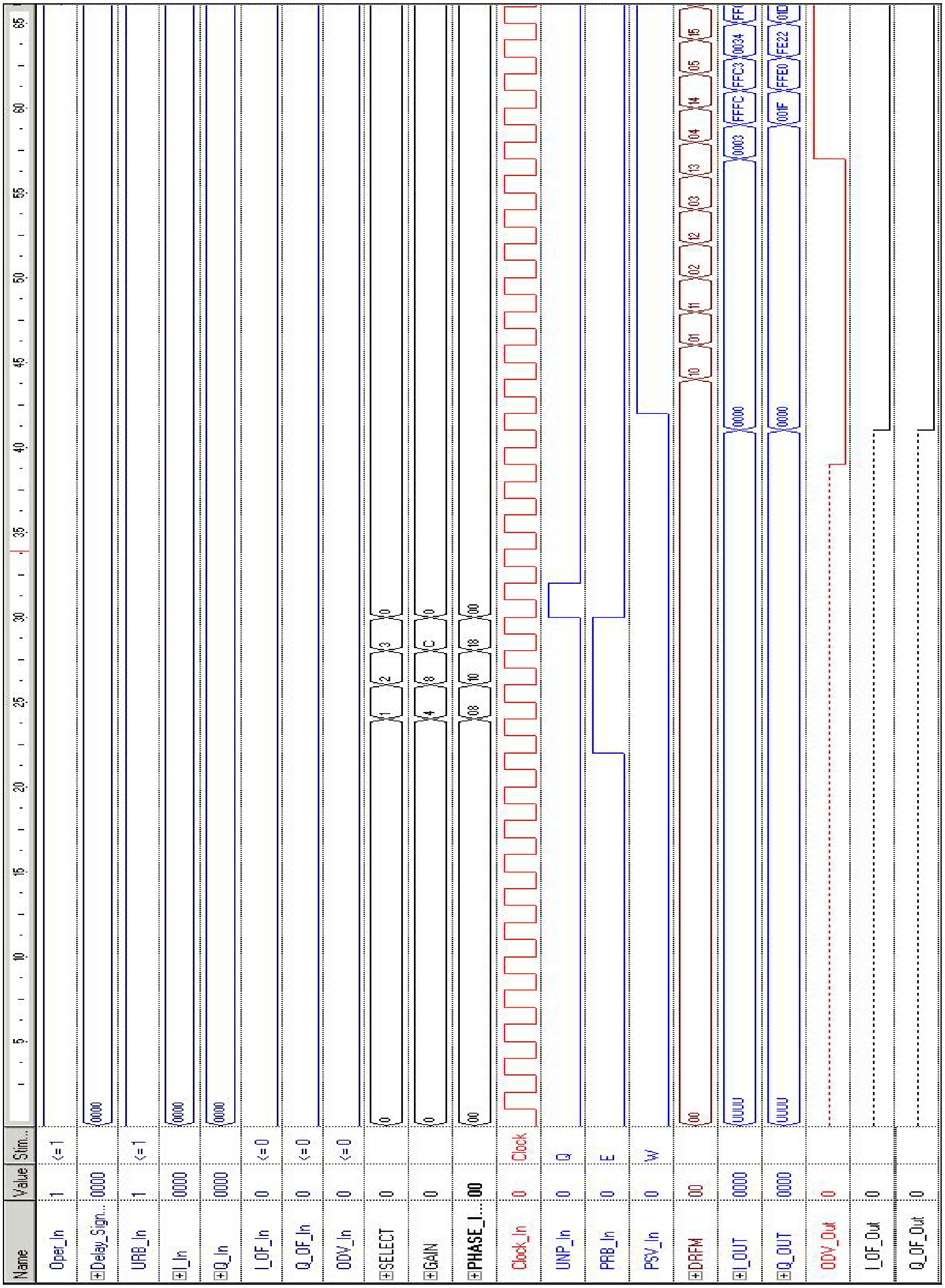
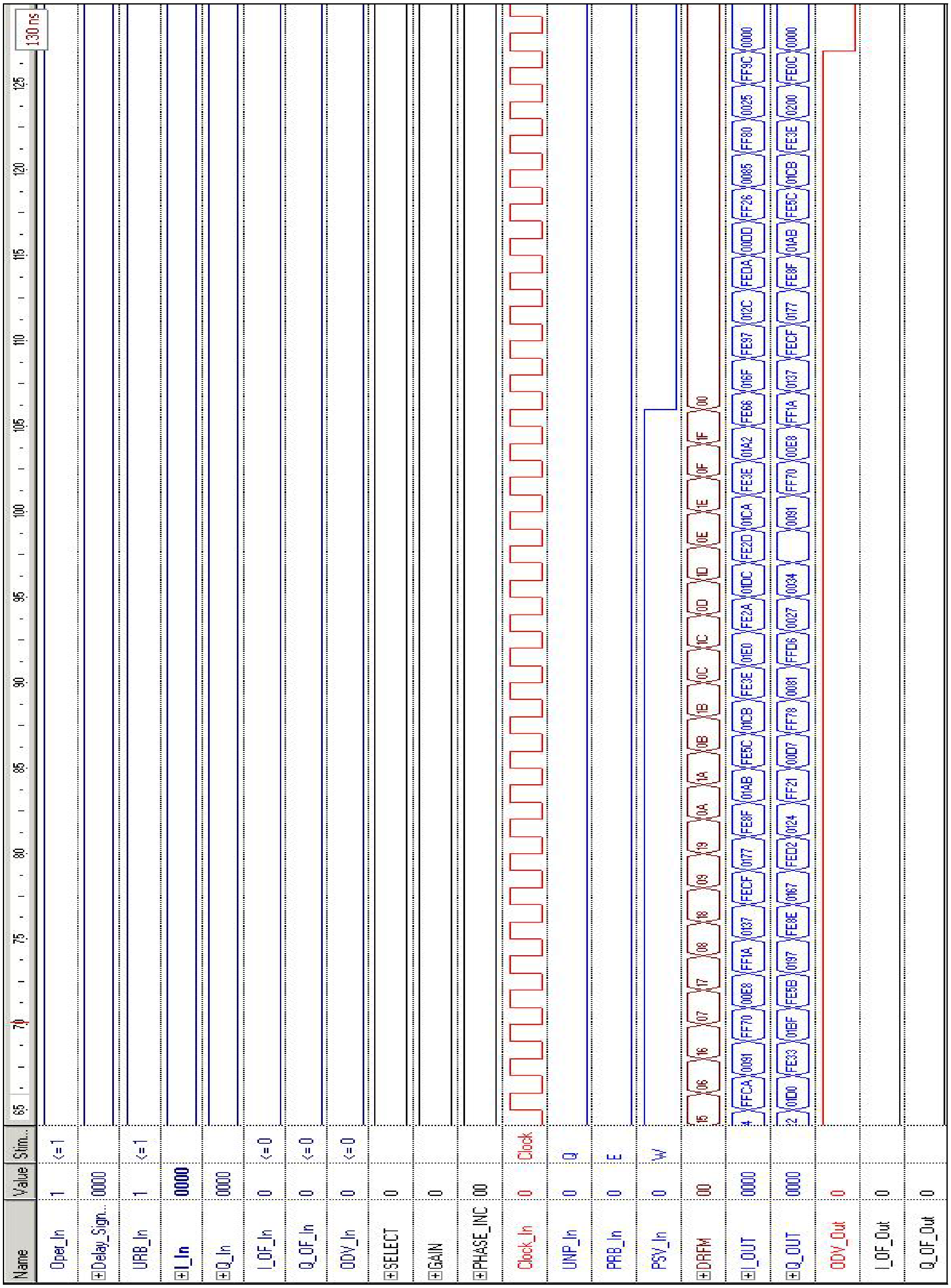


Figure 64.   Simulating Cascaded 4 RBP s


Name
Value
Stim...
Oper_In 1 <=1
Delay_Sign... 0000
URB_In 1 <=1
I_In 0000
Q_In 0000
I_OF_In 0 <=0
Q_OF_In 0 <=0
ODV_In 0 <=0
SELECT 0
GAIN 0
PHASE_INC 00
Clock_In 0 Clock
UNP_In 0 Q
PRB_In 0 E
PSV_In 0 W
DRFM 00
I_OUT 0000
Q_OUT 0000
ODV_Out 0
I_OF_Out 0
Q_OF_Out 0
65
70
75
80
85
90
95
100
105
110
115
120
125
130 ns
15 06 16 07 17 08 18 09 19 0A 1A 0B 1B 0C 1C 0D 1D 0E 1E 0F 1F 00
4 FFCA 0091 FF70 00E8 FF1A 0137 FECF 0177 FE8F 01AB FE5C 01CB FE3E 01E0 FE2A 01DC FE2D 01CA FE3E 01A2 FE66 016F FE97 012C FEDA 00DD FF26 0085 FF80 0025 FF9C 0000
2 01D0 FE33 01BF FE5B 0197 FE8E 0167 FED2 0124 FF21 00D7 FF78 0081 FFD6 0027 0034 0091 FF70 00E8 FF1A 0137 FECF 0177 FE8F 01AB FE5C 01CB FE3E 0200 FE0C 0000

Simulation of Cascaded 4 RBP s, Continued

3. Simulation of 16 RBP s in Series

The simulation algorithm is as follows:

- Set Clock Rate, Clock_In = Stimulator→ Clock → 2ns

- Set Clk_Prg_In, I_In0 through I_In15, Q_In0 through Q_In15, I_OF_In, Q_OF_In, ODV_In by Stimulator→Value→0

- Set Delay='0' in all clock splitting circuits in every RBP using the design browser and by adding signal names into the waveform editor. (Delay signals are in the *entities* DJF_ClockTrue_1 and DJF_ClockComp_1)

- Set Oper_In, URB_In = '1' by Stimulator → Value →1

- Set PSV_In=0, UNP_In=0, PRB_In=0

- Clock RPB for 23 times to clear the pipeline

- Set PRB_In = '1'

- Select the RBP to be programmed using the Sel_In0 through Sel_In8 inputs. Set Gain_In0 through Gain_In3='0',and PInc_In0 through PInc_In4='0' to the desired values for the RBP to be programmed. Clock the RBP once. Repeat for all RBPs.

- Set PRB_In = '0', UNP_In='1'

- Clock RBP once

- Set UNP_In = '0'

- Clock RBP until I_OF_Out, Q_OF_Out and ODV_Out are '0'

- Set PSV_In ='1' and DRFM_In0 through DRFM_In4 to the desired phase sample values, clock RBP, repeat for every DRFM sample value

- Set PSV_In= '0'

- Clock RBP until ODV_Out = '0'

- Watch and record the values for I_Out0 through I_Out15, Q_Out0 through Q_Out15, I_OF_Out (Overflow), Q_OF_Out (Overflow) and ODV_Out

- Compare the results with the C++ outputs.

The waveform used in the simulation of 16 RBP s in series is shown in Figure 65. Table 19 shows the programming coefficients for the RBP s, the simulation results, and the comparison with the C++ outputs.

RBP	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Gain\(Hex)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
PInc(Hex)	00	11	02	13	04	15	06	17	08	19	0A	1B	0C	1D	0E	1F

	Simulation Results				**C++ Outputs**			
Phase Samples (Hex)	**I_Out (Hex)**	**Q_Out (Hex)**	**I_OF _Out**	**Q_OF _Out**	**I_Out (Hex)**	**Q_Out (Hex)**	**I_OF _Out**	**Q_OF _Out**
00	0003	0000	0	0	0003	0000	0	0
01	FFFB	FFFE	0	0	FFFB	FFFE	0	0
02	0009	0003	0	0	0009	0003	0	0
03	FFEE	FFF3	0	0	FFEE	FFF3	0	0
04	0006	0006	0	0	0006	0006	0	0
05	FFE0	FFD3	0	0	FFE0	FFD3	0	0
06	001A	0042	0	0	001A	0042	0	0
07	FFDA	FF4D	0	0	FFDA	FF4D	0	0
08	0000	FF89	0	0	0000	FF89	0	0
09	002C	FF0E	0	0	002C	FF0E	0	0
0A	FFF8	0007	0	0	FFF8	0007	0	0
0B	0112	FE60	0	0	0112	FE60	0	0
0C	FFF5	0006	0	0	FFF5	0006	0	0
0D	0343	FDCB	0	0	0343	FDCB	0	0
0E	FC4A	0189	0	0	FC4A	0189	0	0
0F	0BA9	FDA7	0	0	0BA9	FDA7	0	0
10	0BD9	0000	0	0	0BD9	0000	0	0
11	0BA9	0252	0	0	0BA9	0252	0	0
12	0AEA	0490	0	0	0AEA	0490	0	0
13	09E3	069E	0	0	09E3	069E	0	0
14	0865	0865	0	0	0865	0865	0	0
15	069E	09E3	0	0	069E	09E3	0	0
16	0490	0AEA	0	0	0490	0AEA	0	0
17	0252	0BA9	0	0	0252	0BA9	0	0
18	0000	0BD9	0	0	0000	0BD9	0	0
19	FDA7	0BA9	0	0	FDA7	0BA9	0	0
1A	FB69	0AEA	0	0	FB69	0AEA	0	0
1B	F95B	09E3	0	0	F95B	09E3	0	0
1C	F796	0865	0	0	F796	0865	0	0
1D	F618	069E	0	0	F618	069E	0	0
1E	F50F	0490	0	0	F50F	0490	0	0
1F	F450	0252	0	0	F450	0252	0	0

Table 19. Simulation Results and Comparison for 16 RBP s

	Simulation Results				C++ Outputs			
Phase Samples (Hex)	I_Out (Hex)	Q_Out (Hex)	I_OF _Out	Q_OF _Out	I_Out (Hex)	Q_Out (Hex)	I_OF _Out	Q_OF _Out
-	F41D	0000	0	0	F41D	0000	0	0
-	F455	FDA9	0	0	F455	FDA9	0	0
-	F506	FB66	0	0	F506	FB66	0	0
-	F62A	F968	0	0	F62A	F968	0	0
-	F790	F790	0	0	F790	F790	0	0
-	F97B	F645	0	0	F97B	F645	0	0
-	FB4F	F4CD	0	0	FB4F	F4CD	0	0
-	FDCD	F503	0	0	FDCD	F503	0	0
-	0000	F497	0	0	0000	F497	0	0
-	0226	F542	0	0	0226	F542	0	0
-	0498	F508	0	0	0498	F508	0	0
-	058C	F7B8	0	0	058C	F7B8	0	0
-	0870	F790	0	0	0870	F790	0	0
-	06A0	FB90	0	0	06A0	FB90	0	0
-	0EA0	F9E0	0	0	0EA0	F9E0	0	0

Simulation Results and Comparison for 16 RBP s, Continued

After verification of 16 RBP s' functionality, a test was conducted to test the use of only 13 RBP s in a cascade of 16 RBP s. Some modifications were made to the simulation algorithm. URB_In was set to "0" for RBP s 13, 14 and 15. Thus, the 16 RBP cascade acted like 13 RBP s connected sequentially.

The simulation results and C++ outputs for 13 RBP s cascaded is given in Table 20, while the waveform is shown in Figure 66. The simulations with a single RBP, 4 RBP s, 16 RBPs and 13 RBPs serially connected were tested and the DIS was verified.

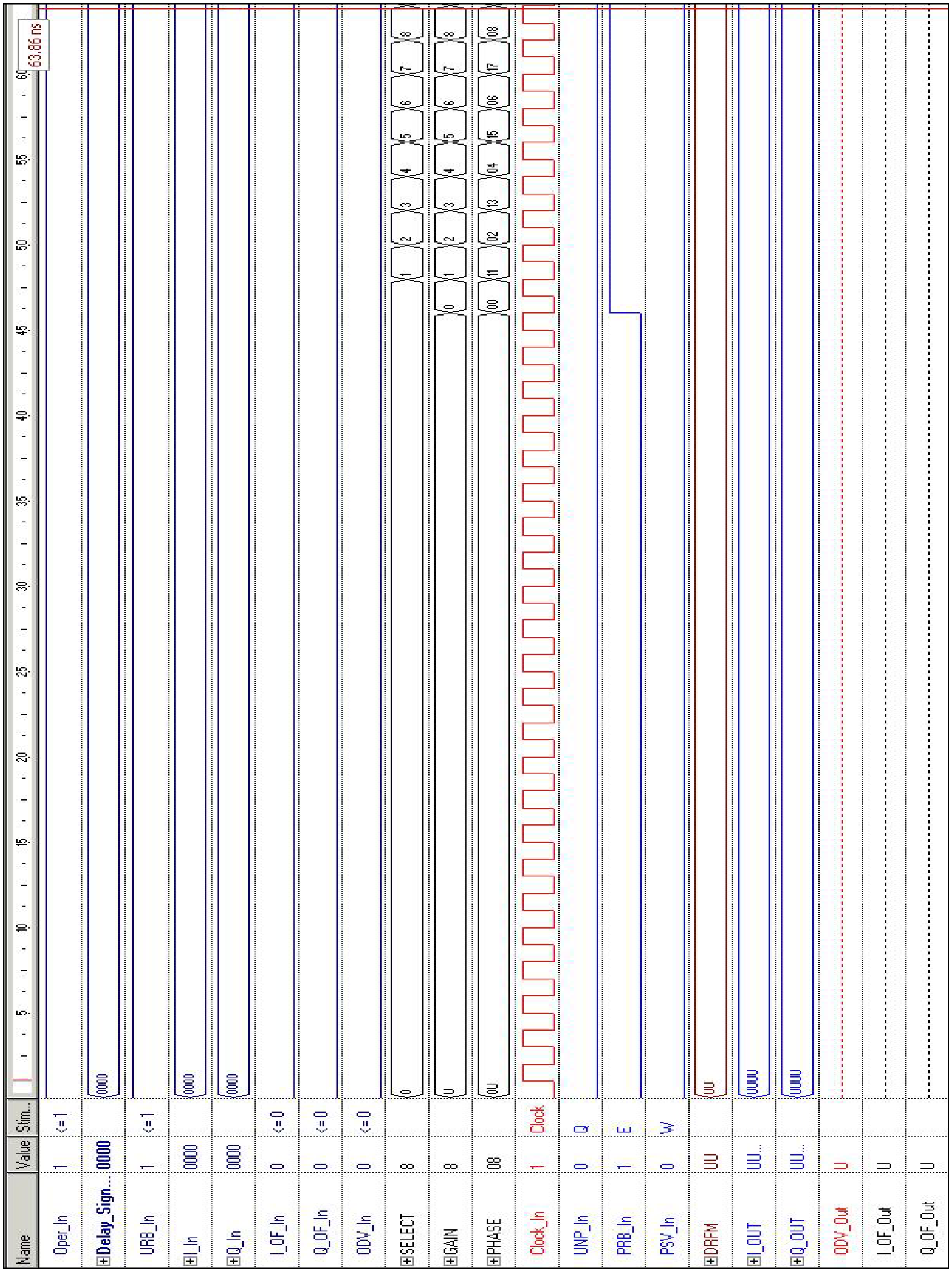
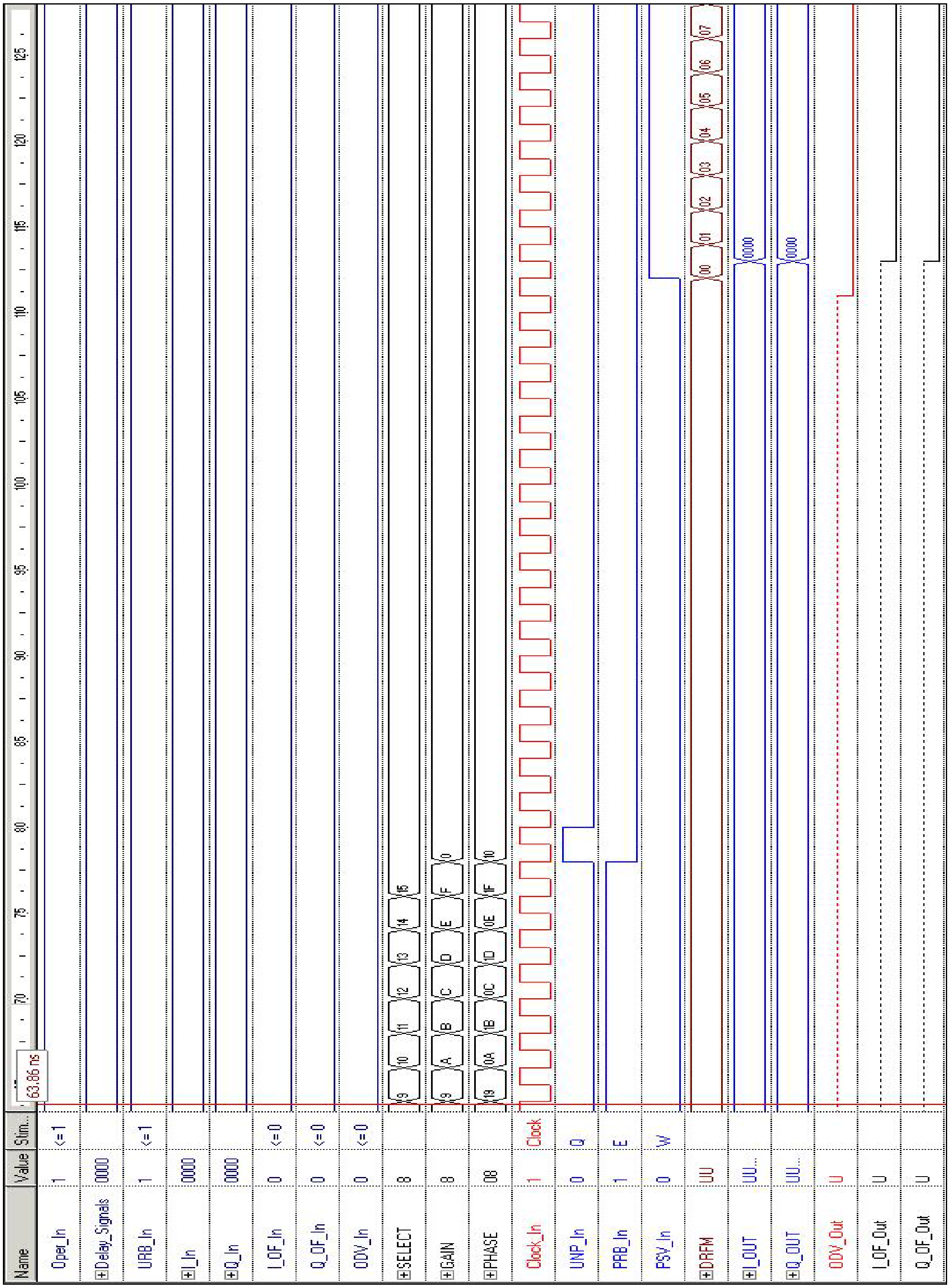
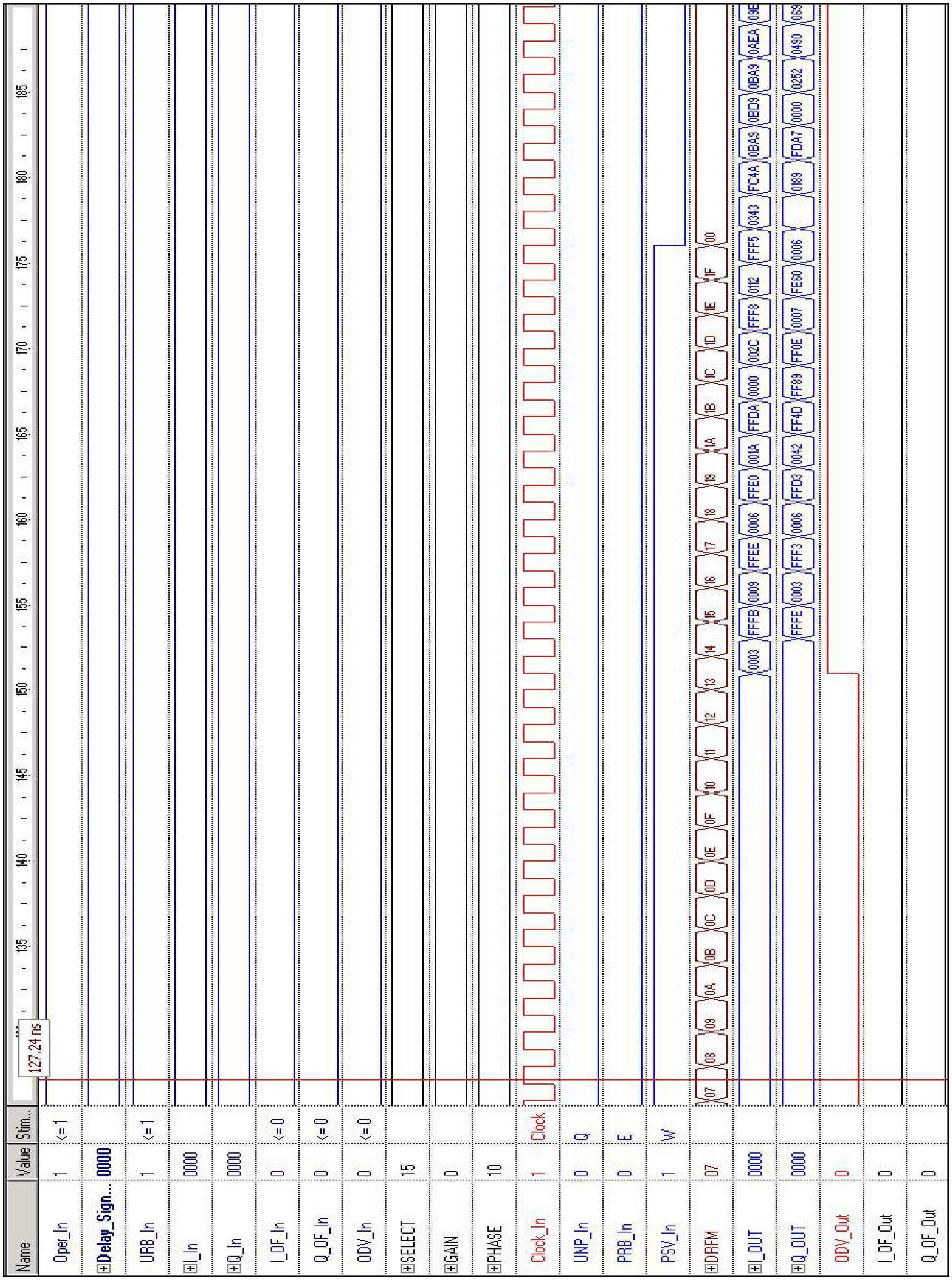


Figure 65. Simulation of Cascaded 16 RBP s

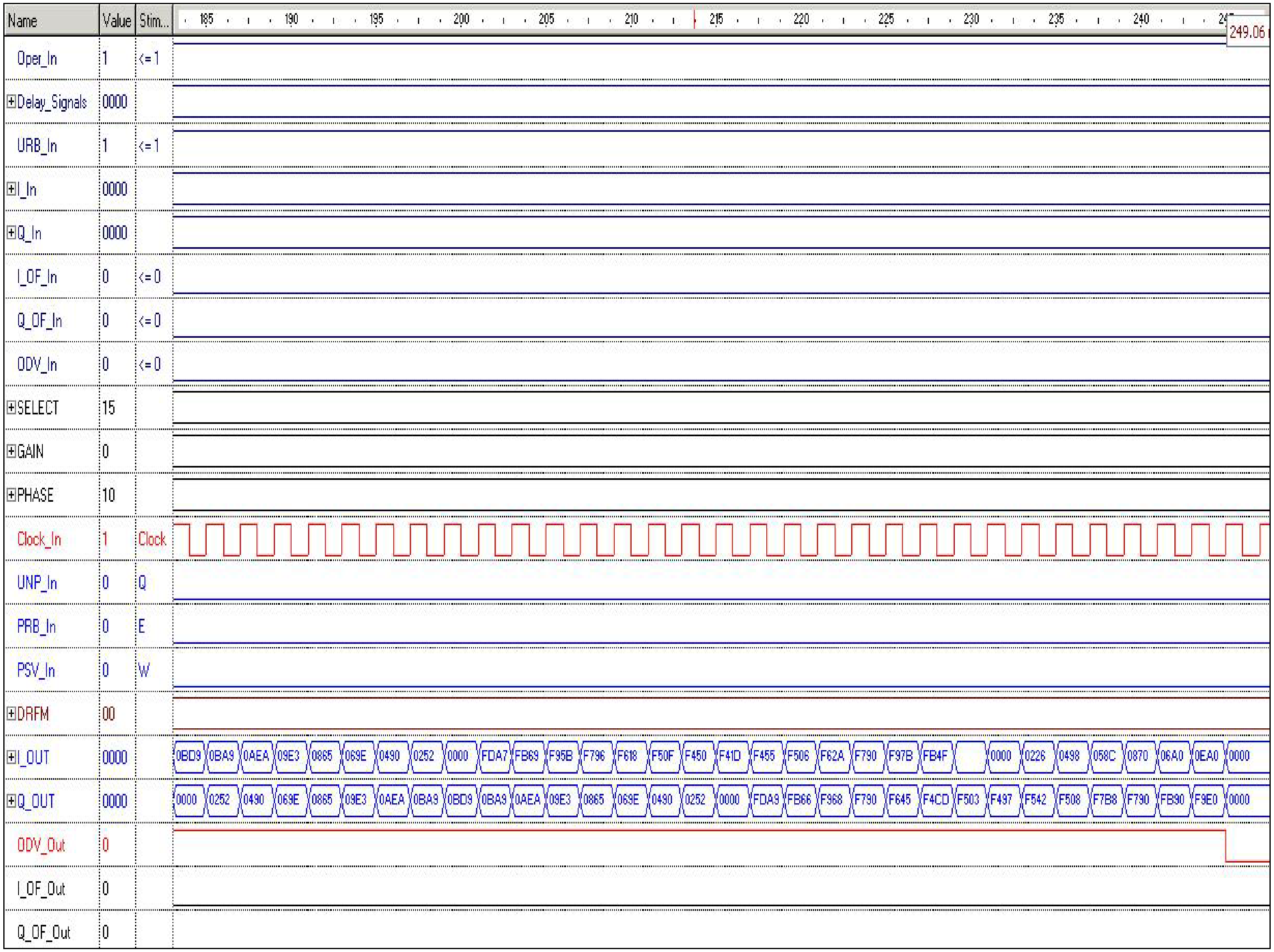


Name	Value	Stim...
Oper_In	1	<=1
⊞Delay_Signals	0000	
URB_In	1	<=1
⊞I_In	0000	
⊞Q_In	0000	
I_OF_In	0	<=0
Q_OF_In	0	<=0
ODV_In	0	<=0
⊞SELECT	8	
⊞GAIN	8	
⊞PHASE	08	
Clock_In	1	Clock
UNP_In	0	Q
PRB_In	1	E
PSV_In	0	W
⊞DRFM	UU	
⊞I_OUT	UU...	
⊞Q_OUT	UU...	
ODV_Out	U	
I_OF_Out	U	
Q_OF_Out	U	

Simulation of Cascaded 16 RBP s, Continued



Simulation of Cascaded 16 RBP s, Continued



Name	Value	Stim...
Oper_In	1	<=1
⊞Delay_Signals	0000	
URB_In	1	<=1
⊞I_In	0000	
⊞Q_In	0000	
I_OF_In	0	<=0
Q_OF_In	0	<=0
ODV_In	0	<=0
⊞SELECT	15	
⊞GAIN	0	
⊞PHASE	10	
Clock_In	1	Clock
UNP_In	0	Q
PRB_In	0	E
PSV_In	0	W
⊞DRFM	00	
⊞I_OUT	0000	
⊞Q_OUT	0000	
ODV_Out	0	
I_OF_Out	0	
Q_OF_Out	0	

185
190
195
200
205
210
215
220
225
230
235
240
249.06
0BD9 0BA9 0AEA 09E3 0865 069E 0490 0252 0000 FDA7 FB69 F95B F796 F618 F50F F450 F41D F455 F506 F62A F790 F97B FB4F 0000 0226 0498 058C 0870 06A0 0EA0 0000
0000 0252 0490 069E 0865 09E3 0AEA 0BA9 0BD9 0BA9 0AEA 09E3 0865 069E 0490 0252 0000 FDA9 FB66 F968 F790 F645 F4CD F503 F497 F542 F508 F7B8 F790 FB90 F9E0 0000

Simulation of Cascaded 16 RBP s, Continued

RBP	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Gain\(Hex)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
PInc(Hex)	00	11	02	13	04	15	06	17	08	19	0A	1B	0C	1D	0E	1F

	Simulation Results				C++ Outputs			
Phase Samples (Hex)	I_Out (Hex)	Q_Out (Hex)	I_OF _Out	Q_OF _Out	I_Out (Hex)	Q_Out (Hex)	I_OF _Out	Q_OF _Out
00	0003	0000	0	0	0003	0000	0	0
01	FFFB	FFFE	0	0	FFFB	FFFE	0	0
02	0009	0003	0	0	0009	0003	0	0
03	FFEE	FFF3	0	0	FFEE	FFF3	0	0
04	0006	0006	0	0	0006	0006	0	0
05	FFE0	FFD3	0	0	FFE0	FFD3	0	0
06	001A	0042	0	0	001A	0042	0	0
07	FFDA	FF4D	0	0	FFDA	FF4D	0	0
08	0000	FF89	0	0	0000	FF89	0	0
09	002C	FF0E	0	0	002C	FF0E	0	0
0A	FFF8	0007	0	0	FFF8	0007	0	0
0B	0112	FE60	0	0	0112	FE60	0	0
0C	FFF5	0006	0	0	FFF5	0006	0	0
0D	FFF3	0003	0	0	FFF3	0003	0	0
0E	FFF2	0001	0	0	FFF2	0001	0	0
0F	FFF1	FFFF	0	0	FFF1	FFFF	0	0
10	FFF1	0000	0	0	FFF1	0000	0	0
11	FFF1	FFFA	0	0	FFF1	FFFA	0	0
12	FFF2	FFF8	0	0	FFF2	FFF8	0	0
13	FFF3	FFF6	0	0	FFF3	FFF6	0	0
14	FFF5	FFF5	0	0	FFF5	FFF5	0	0
15	FFF6	FFF3	0	0	FFF6	FFF3	0	0
16	FFF8	FFF2	0	0	FFF8	FFF2	0	0
17	FFFA	FFF1	0	0	FFFA	FFF1	0	0
18	0000	FFF1	0	0	0000	FFF1	0	0
19	FFFF	FFF1	0	0	FFFF	FFF1	0	0
1A	0001	FFF2	0	0	0001	FFF2	0	0
1B	0003	FFF3	0	0	0003	FFF3	0	0
1C	0006	FFF5	0	0	0006	FFF5	0	0
1D	0008	FFF6	0	0	0008	FFF6	0	0
1E	0007	FFF8	0	0	0007	FFF8	0	0
1F	0008	FFFA	0	0	0008	FFFA	0	0
-	0005	0000	0	0	0005	0000	0	0
-	000D	0001	0	0	000D	0001	0	0
-	FFFE	FFFE	0	0	FFFE	FFFE	0	0
-	001A	0010	0	0	001A	0010	0	0
-	0000	0000	0	0	0000	0000	0	0
-	0023	0035	0	0	0023	0035	0	0
-	FFE7	FFC5	0	0	FFE7	FFC5	0	0
-	0025	00BB	0	0	0025	00BB	0	0
-	0000	007F	0	0	0000	007F	0	0
-	FFCE	00FA	0	0	FFCE	00FA	0	0
-	0000	0000	0	0	0000	0000	0	0
-	FEE4	01A8	0	0	FEE4	01A8	0	0

Table 20.   Simulation Results and Comparison for 13 RBP s

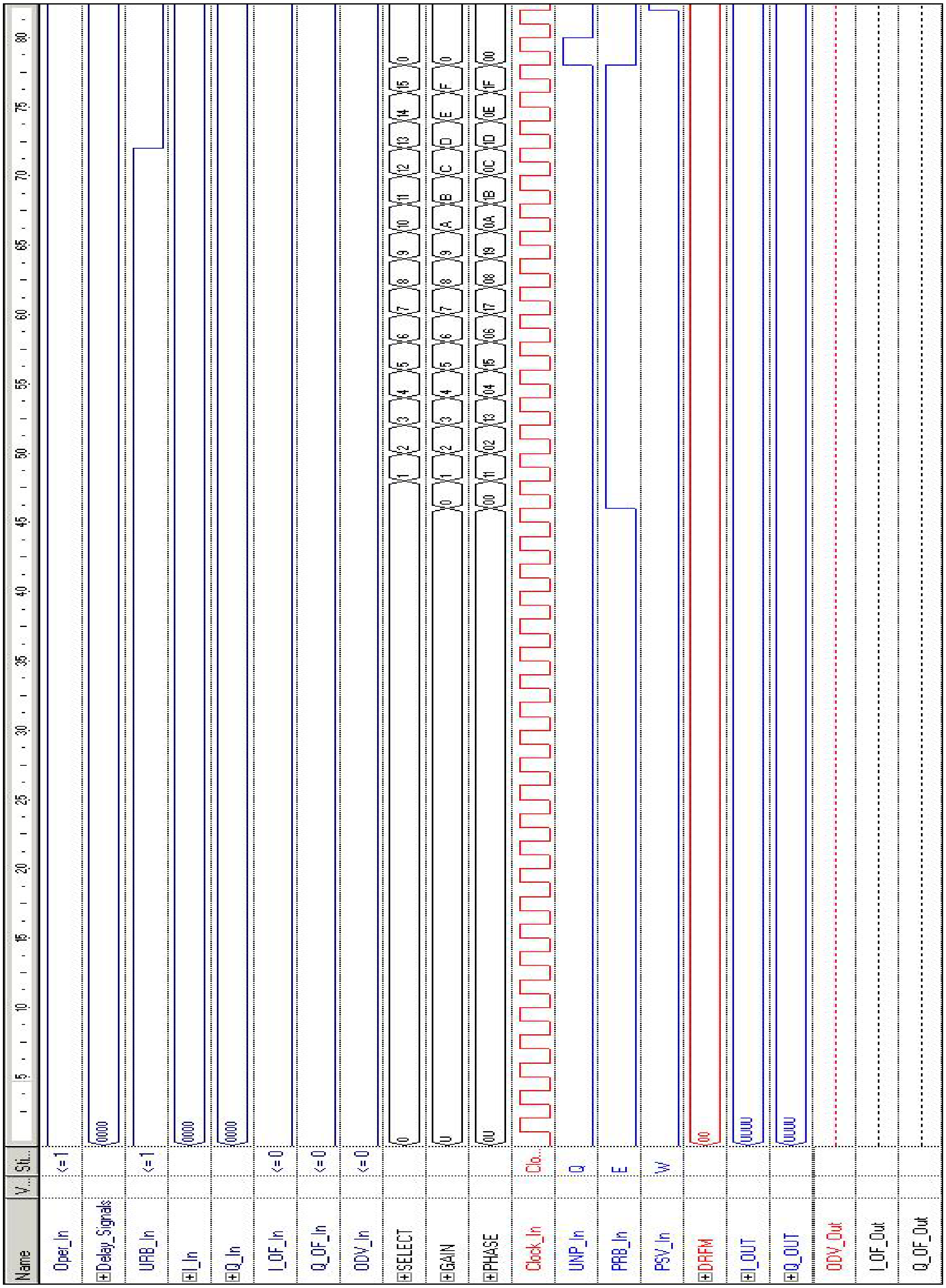
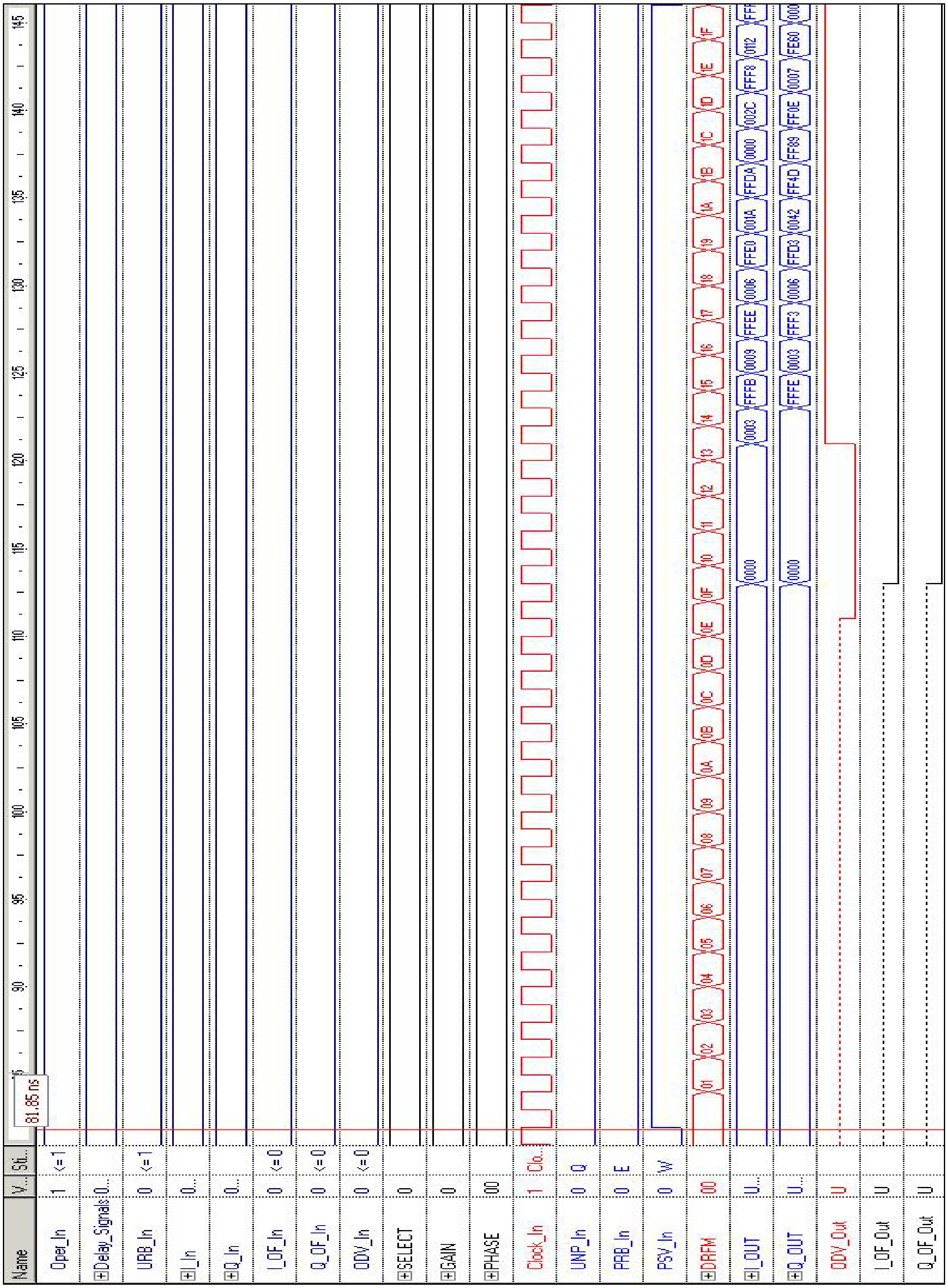
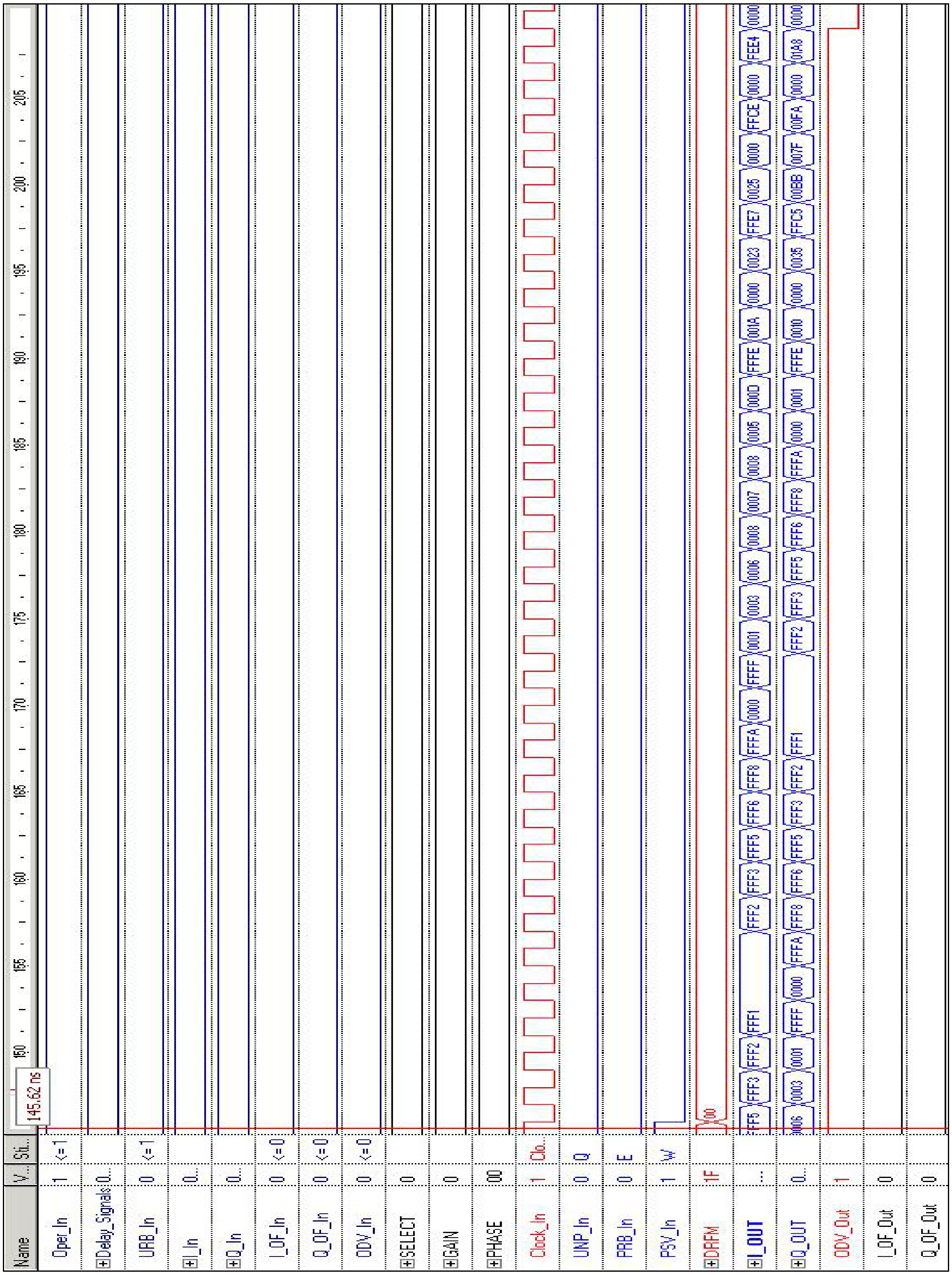


Figure 66. Simulation of Cascaded 13 RBP s



Simulation of Cascaded 13 RBP s, Continued



Simulation of Cascaded 13 RBP s, Continued

4. Simulation of the Self Test Circuit

The Self Test Circuit can generate 4095 random phase sample values for which the correct target signature is known. It is utilized for self-test of the DIS. The simulation algorithm to test the Self Test Circuit is as follows:

- Set Clock Rate, Clock_In = Stimulator→ Clock → 2ns
- Set Start_Self_Test = ‘0’
- Clock the circuit once
- Set Start_Self_Test =’1’
- Clock the circuit for 4097 times to get all random test vectors generated. There is a 3-clock-cycle-delay between Start_Self_Test signal’s “low” to “high” transition and the PSV output “low” to “high” transition.

The beginning and the end of the simulation shown on the waveform editors are in Figure 67 and Figure 68, while the resultant test vectors are listed in Appendix A.

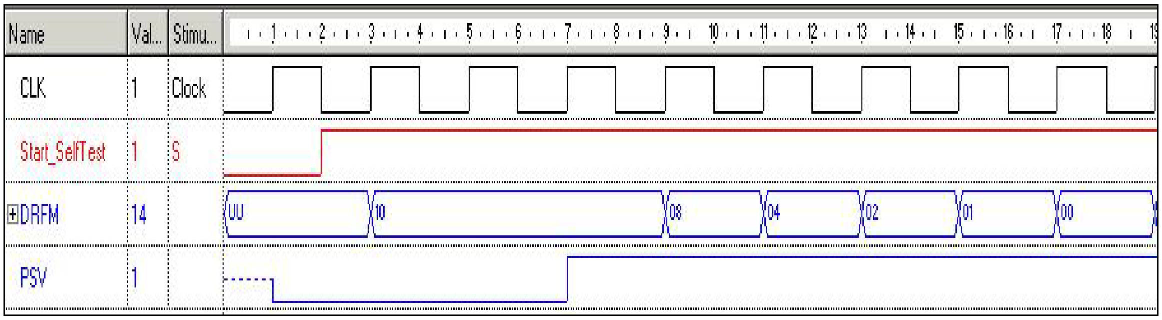


Figure 67. Simulation of Self Test Circuit, Beginning

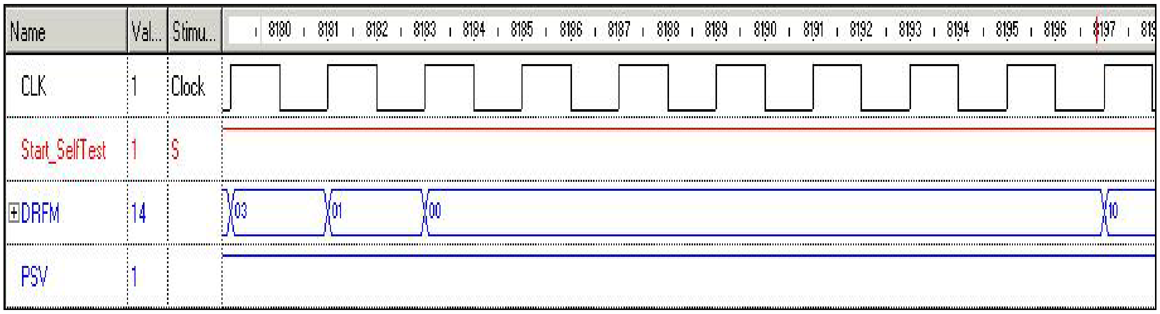


Figure 68. Simulation of Self Test Circuit, Ending

5. Simulation of the Phase Extraction Circuit

An exhaustive test was conducted to verify all possible input/output combinations. The simulation algorithm used is listed below.

- Set Clock Rate, Clock_In = Stimulator→ Clock → 0.5ns
- Set Load = ‘1’ using Stimulator → Value → ‘1’ (In the DIS, this input is tied to Vdd, logic ‘1’)
- Set I_In0 through I_In7 and Q_In0 through Q_In7 = ‘0’ and PSV_In = ‘1’.
- Clock the circuit for 16 times to initialize the pipeline registers.
- Set PSV_In=’0’ and clock the circuit for 16 times. Observe the “high” to low transition on PSV_Out.
- Set PSV_In=’1’
- Set I_In0 through I_In7 and Q_In0 through Q_In7 to desired value. Clock the circuit. Repeat for all possible input values. (To apply all possible inputs it is very helpful to use Stimulator → Clock.)
- Set PSV_In = ‘0’, clock the circuit for 16 times to empty the pipeline.
- Document the outputs Phase_Out0 through Phase_Out4 and compare with C++ results.

The comparison between the simulation results and the C++ outputs, accomplished by Prof, Fouts, showed that the phase extractor works correctly. Initialization of the simulation is shown in a waveform editor in Figure 69, while Figure 70 points to the end of the simulation and clearing of the pipeline.

Since inputs range between –128 and 127 for both I and Q values, 65,536 different input combinations were used. Some of the values for comparison are shown in Table 21.

I_Value (DEC)	Q_Value (DEC)	Simulation Result Phase (HEX)	C++ Result Phase (HEX)	I_Value (DEC)	Q_Value (DEC)	Simulation Result Phase (HEX)	C++ Result Phase (HEX)
-128	-128	20	20	0	-128	24	24
-128	-105	19	19	0	-1	24	24
-128	-60	18	18	0	0	0	0
-128	-42	17	17	0	1	8	8
-128	-9	16	16	0	127	8	8
-128	0	16	16	1	-128	24	24
-128	10	15	15	1	0	0	0
-128	43	14	14	1	1	4	4
-128	61	13	13	1	2	5	5
-128	106	12	12	1	3	7	7
-128	127	12	12	1	13	8	8
-127	-128	20	20	1	127	8	8
-127	-105	19	19	127	-128	28	28
-127	-60	18	18	127	-105	29	29
-127	-42	17	17	127	-60	30	30
-127	-9	16	16	127	-42	31	31
-127	10	15	15	127	-9	0	0
-127	43	14	14	127	10	1	1
-127	61	13	13	127	43	2	2
-127	106	12	12	127	61	3	3
-127	127	12	12	127	106	4	4
-100	0	16	16	100	0	0	0
-100	120	12	12	100	120	4	4
-100	121	11	11	100	121	5	5
-99	-128	21	21	101	-128	27	27
-19	-128	23	23	19	-128	25	25
-19	-56	22	22	19	-57	26	26
-19	-40	21	21	19	-40	27	27
-19	0	16	16	19	0	0	0
-19	127	9	9	19	127	7	7

Table 21. Comparison of Simulation Results and C++ Outputs for Phase Extractor

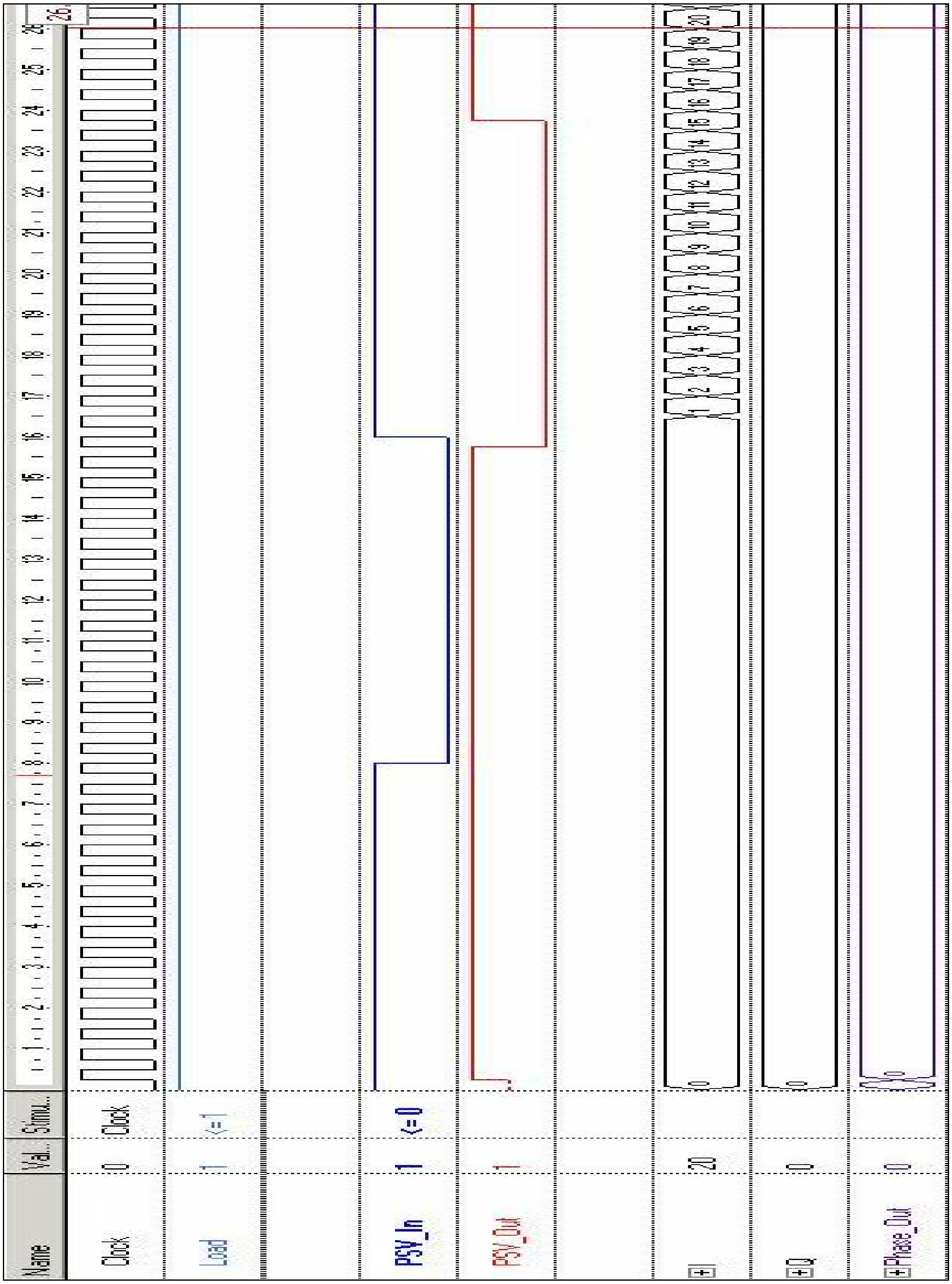


Figure 69. Simulation of Phase Extraction Circuit, Initialization

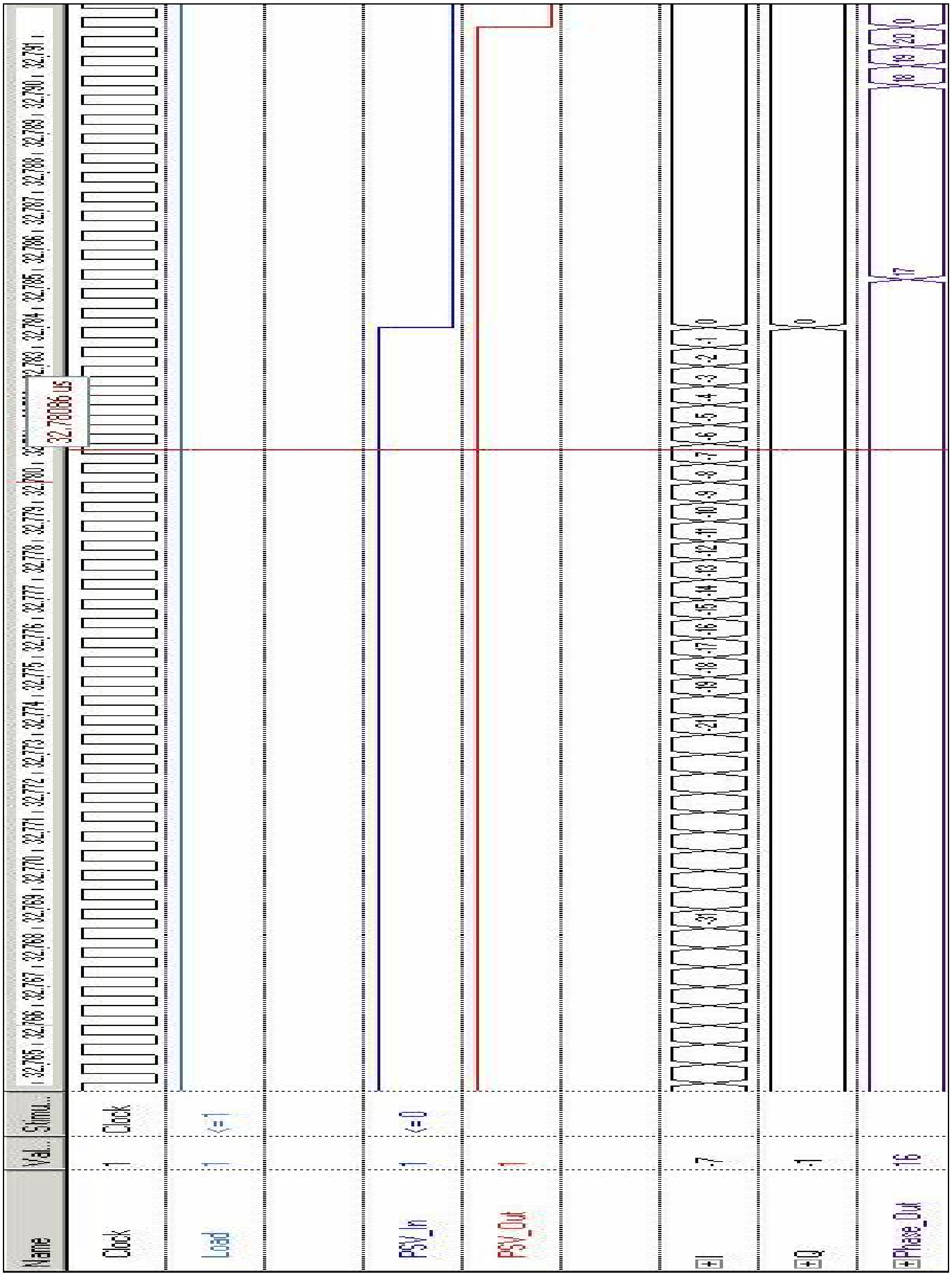


Figure 70. Simulation of Phase Extraction Circuit, Ending

6. Simulation of Path 1 – Off-Chip Phase Sample Values to RBP s

The simulation algorithm for Path 1, the flow from the off chip phase sample values to the four RBP s connected serially, is given below.

- Set Clock_In = Stimulator → Clock → 2ns.
- Set Delay signals inside RBP s ='0'
- Set Clock_Prog_In = '0', URB_In='0'
- Set ODV_In, PRB_In and UNP_In ='0'
- Set I0_0 through I0_5 ='0'
- Set I_In_0 through I_In_15, Q_In_0 through Q_In_15, I_OF_In, Q_OF_In, I0 through I7, Q0 through Q7, I1_0 through I1_5 ='0'
- Set Off_Chip_Count0 through Off_Chip_Count11, I/Q_Valid_In, Off_Chip_4to1MuxSLCT0, Off_Chip_4to1MuxSLCT1,Start_SelfTest ='0'
- Set Off_Chip_Oper/Maint_MuxIO ='1'
- Set Off_Chip_Oper/Maint_MuxSel ='0'
- Clock the DIS for 23 times to clear the pipeline inside RBP s.
- Set PRB_In ='1'
- Set Sel_In0 through Sel_In7 to the desired RBP number; set Gain_In_0 through Gain_In_3 and Phase_In_0 through Phase_In_4 to the proper coefficient values. Clock the DIS once. Repeat for every RBP to be programmed.
- Set PRB = '0', UNP='1', clock the DIS once
- Clock the DIS until ODV_Out becomes "low"
- Set I0_5 ='1' (This input is actually PSV_In to the RBP s after being steered by the 6-bit 4-to-1 multiplexer)
- Set I0_0 through I0_4 to the desired phase sample value. Clock the DIS once. Repeat for every off chip phase sample value.

- Set I0_5=’0’

- Clock the DIS for 11 times to empty the pipeline, until ODV_Out becomes “low”

- Observe MUX_Out0 through MUX_Out5 to verify the inputs are steered into the RBP s from the 6-bit 4-to-1 multiplexer.

- Observe ODV_Out, I_Out_0 through I_Out_15, Q_Out_0 through Q_Out_15, I_OF_Out and Q_OF_Out.

- Compare the results with the C++ simulation outputs.

The waveform editor used to simulate the DIS for the first data path is given in Figures 71, 72 and 73, showing initialization, input phase sample values, and the end of the simulation, one after the other.

Table 22 shows the RBP programming coefficients, phase sample input values to the first data path, signal values probed at the output of 6-bit 4-to-1 multiplexer and outputs of the DIS.

7. Simulation of Path 2 – Off-Chip Phase Sample Alternate Path

The simulation algorithm for Path 2, the flow from the alternate off chip phase sample values to the four RBP s connected serially, is very similar to the simulation of Path 1 and is given below.

- Set Clock_In = Stimulator → Clock → 2ns.

- Set Delay signals inside RBP s =’0’

- Set Clock_Prog_In = ‘0’, URB_In=’0’

- Set ODV_In, PRB_In and UNP_In =’0’

- Set I1_0 through I1_5 =’0’

- Set I_In_0 through I_In_15, Q_In_0 through Q_In_15, I_OF_In, Q_OF_In, I0 through I7, Q0 through Q7, I0_0 through I0_5 =’0’

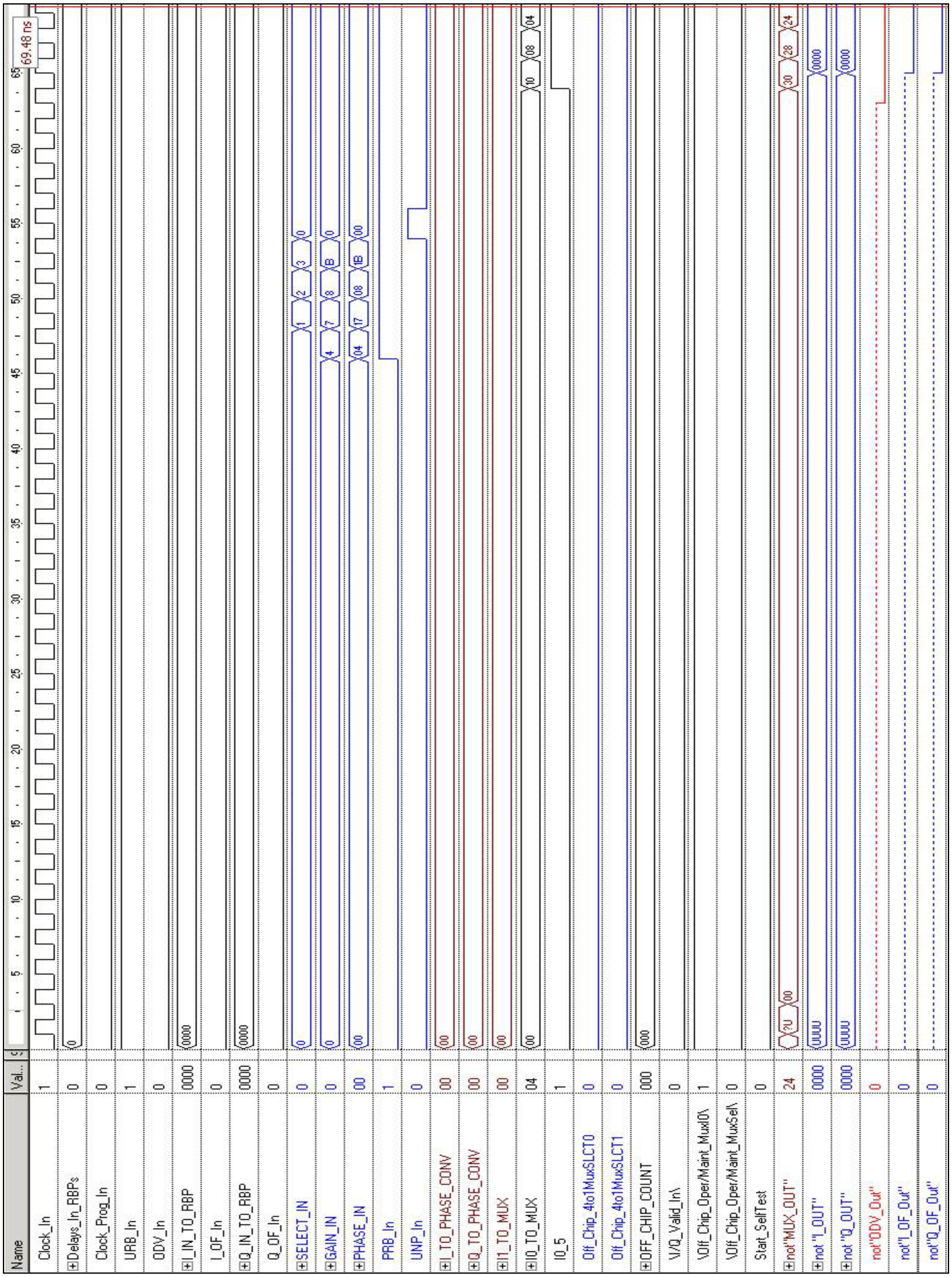


Figure 71. Simulation of the DIS – Path 1, Initialization

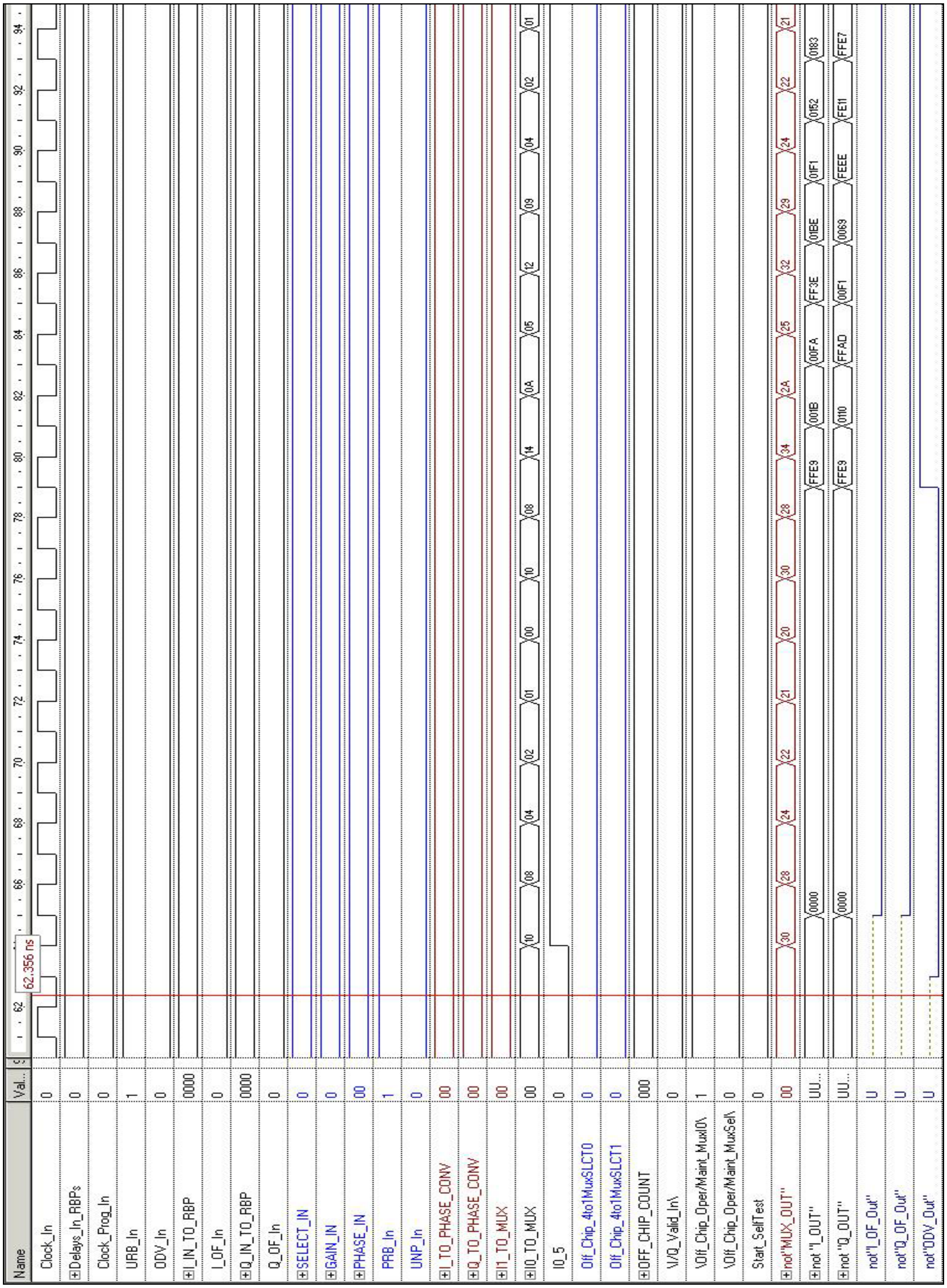


Figure 72. Simulation of the DIS – Path 1, Inputting Phase Samples

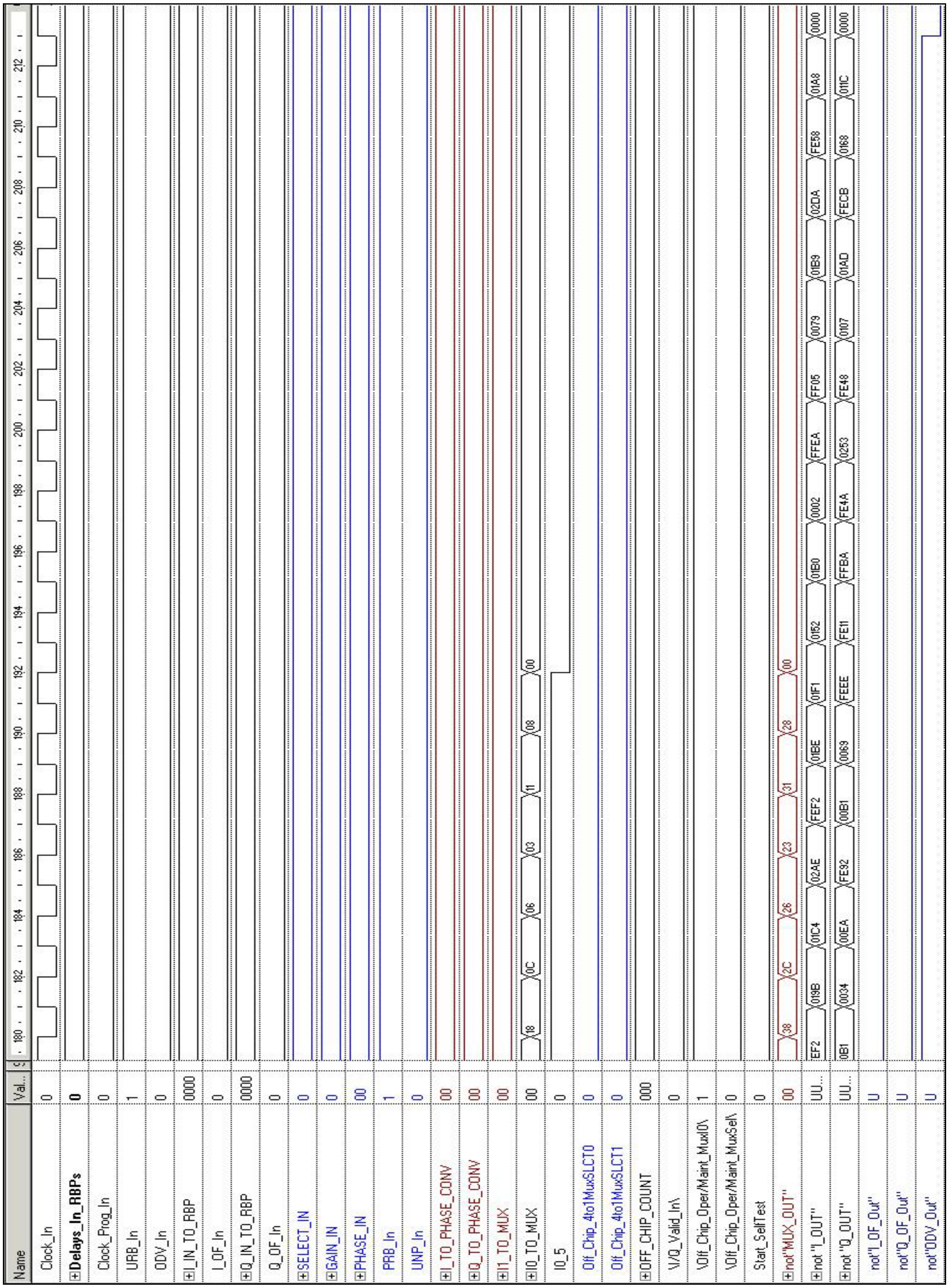


Figure 73. Simulation of the DIS – Path 1, Ending

RBP	0	1	2	3
Gain\(Hex)	04	07	08	0B
PInc(Hex)	04	17	08	1B

	Simulation Results					C++ Outputs				
I0 Phase Samples (Hex)	MUX Out (Hex)	I_Out (Hex)	Q_Out (Hex)	I_OF _Out	Q_OF _Out	MUX Out (Hex)	I_Out (Hex)	Q_Out (Hex)	I_OF _Out	Q_OF _Out
10	10	FFE9	FFE9	0	0	10	FFE9	FFE9	0	0
08	08	001B	0110	0	0	08	001B	0110	0	0
04	04	00FA	FFAD	0	0	04	00FA	FFAD	0	0
02	02	FF3E	00F1	0	0	02	FF3E	00F1	0	0
01	01	01BE	0069	0	0	01	01BE	0069	0	0
00	00	01F1	FEEE	0	0	00	01F1	FEEE	0	0
10	10	0152	FE11	0	0	10	0152	FE11	0	0
08	08	0183	FFE7	0	0	08	0183	FFE7	0	0
14	14	0216	FDC6	0	0	14	0216	FDC6	0	0
0A	0A	FDF8	0288	0	0	0A	FDF8	0288	0	0
05	05	02C8	0140	0	0	05	02C8	0140	0	0
12	12	FE78	FF79	0	0	12	FE78	FF79	0	0
09	09	009A	02D6	0	0	09	009A	02D6	0	0
04	04	0312	FFE4	0	0	04	0312	FFE4	0	0
02	02	FEB3	0058	0	0	02	FEB3	0058	0	0
01	01	017E	00B5	0	0	01	017E	00B5	0	0
10	10	01C4	FEC1	0	0	10	01C4	FEC1	0	0
08	08	01B6	0032	0	0	08	01B6	0032	0	0
14	14	0262	FE06	0	0	14	0262	FE06	0	0
0A	0A	FDF8	0288	0	0	0A	FDF8	0288	0	0
15	15	02D5	0101	0	0	15	02D5	0101	0	0
0A	0A	FCFF	010B	0	0	0A	FCFF	010B	0	0
05	05	0244	01D5	0	0	05	0244	01D5	0	0
12	12	FE70	FF15	0	0	12	FE70	FF15	0	0
09	09	009A	02D6	0	0	09	009A	02D6	0	0
04	04	0312	FFE4	0	0	04	0312	FFE4	0	0
02	02	FEB3	0058	0	0	02	FEB3	0058	0	0
11	11	015B	0080	0	0	11	015B	0080	0	0
08	08	01C4	00EA	0	0	08	01C4	00EA	0	0
04	04	02AE	FE92	0	0	04	02AE	FE92	0	0
12	12	FED9	0076	0	0	12	FED9	0076	0	0
09	09	012E	0254	0	0	09	012E	0254	0	0
14	14	030A	FF41	0	0	14	030A	FF41	0	0
1A	1A	FDA8	01D6	0	0	1A	FDA8	01D6	0	0
1D	1D	00BA	010F	0	0	1D	00BA	010F	0	0
0E	0E	FD74	FFBB	0	0	0E	FD74	FFBB	0	0
17	17	FFA6	FF46	0	0	17	FFA6	FF46	0	0
0B	0B	FEDD	FE31	0	0	0B	FEDD	FE31	0	0
15	15	00CA	0229	0	0	15	00CA	0229	0	0

Table 22. Comparison of Simulation Results and C++ Outputs for Path 1

	Simulation Results					C++ Outputs				
I0 Phase Samples (Hex)	MUX Out (Hex)	I_Out (Hex)	Q_Out (Hex)	I_OF _Out	Q_OF _Out	MUX Out (Hex)	I_Out (Hex)	Q_Out (Hex)	I_OF _Out	Q_OF _Out
0A	0A	FD25	FFD8	0	0	0A	FD25	FFD8	0	0
05	05	01EC	0201	0	0	05	01EC	0201	0	0
02	02	FE89	FF50	0	0	02	FE89	FF50	0	0
11	11	0107	00B6	0	0	11	0107	00B6	0	0
18	18	01F9	0121	0	0	18	01F9	0121	0	0
0C	0C	009A	FED7	0	0	0C	009A	FED7	0	0
06	06	FF9E	0213	0	0	06	FF9E	0213	0	0
03	03	FF05	FE48	0	0	03	FF05	FE48	0	0
11	11	0079	0107	0	0	11	0079	0107	0	0
08	08	01B9	01AD	0	0	08	01B9	01AD	0	0
04	04	02DA	FEEA	0	0	04	02DA	FEEA	0	0
-	-	FEF2	00B1	0	0	-	FEF2	00B1	0	0
-	-	019B	0034	0	0	-	019B	0034	0	0
-	-	01C4	00EA	0	0	-	01C4	00EA	0	0
-	-	02AE	FE92	0	0	-	02AE	FE92	0	0
-	-	FEF2	00B1	0	0	-	FEF2	00B1	0	0
-	-	01BE	0069	0	0	-	01BE	0069	0	0
-	-	01F1	FEEE	0	0	-	01F1	FEEE	0	0
-	-	0152	FE11	0	0	-	0152	FE11	0	0
-	-	01B0	FFBA	0	0	-	01B0	FFBA	0	0
-	-	0002	FE4A	0	0	-	0002	FE4A	0	0
-	-	FFEA	0253	0	0	-	FFEA	0253	0	0
-	-	FF05	FE48	0	0	-	FF05	FE48	0	0
-	-	0079	0107	0	0	-	0079	0107	0	0
-	-	01B9	01AD	0	0	-	01B9	01AD	0	0
-	-	02DA	FECB	0	0	-	02DA	FECB	0	0
-	-	FE58	0168	0	0	-	FE58	0168	0	0
-	-	01A8	011C	0	0	-	01A8	011C	0	0

Comparison of Simulation Results and C++ Outputs for Path 1, Continued

- Set Off_Chip_Count0 through Off_Chip_Count11, I/Q_Valid_In, Set Off_Chip_4to1MuxSLCT1, Start_SelfTest =’0’
- Set Off_Chip_4to1MuxSLCT0 = ‘1’
- Set Off_Chip_Oper/Maint_MuxIO =’1’
- Set Off_Chip_Oper/Maint_MuxSel =’0’
- Clock the DIS for 23 times to clear the pipeline inside RBP s.
- Set PRB_In =’1’

- Set Sel_In0 through Sel_In7 to the desired RBP number; set Gain_In_0 through Gain_In_3 and Phase_In_0 through Phase_In_4 to the proper coefficient values. Clock the DIS once. Repeat for every RBP to be programmed.

- Set PRB = ‘0’, UNP=’1’, clock the DIS once

- Clock the DIS until ODV_Out becomes “low”

- Set I1_5 =’1’ (This input is actually PSV_In to the RBP s after being steered by the 6-bit 4-to-1 multiplexer)

- Set I1_0 through I1_4 to the desired phase sample value. Clock the DIS once. Repeat for every off chip phase sample value.

- Set I1_5=’0’

- Clock the DIS for 11 times to empty the pipeline, until ODV_Out becomes “low”

- Observe MUX_Out0 through MUX_Out5 to verify the inputs are steered into the RBP s from the 6-bit 4-to-1 multiplexer.

- Observe ODV_Out, I_Out_0 through I_Out_15, Q_Out_0 through Q_Out_15, I_OF_Out and Q_OF_Out.

- Compare the results with the C++ simulation outputs.

The waveform editor used to simulate the DIS for the second data path is given in Figures 74, 75 and 76, showing initialization, input phase sample values and the end of simulation, one after the other.

Table 23 shows the RBP programming coefficients, phase sample input values to the second data path, signal values probed at the output of 6-bit 4-to-1 multiplexer and outputs of the DIS.

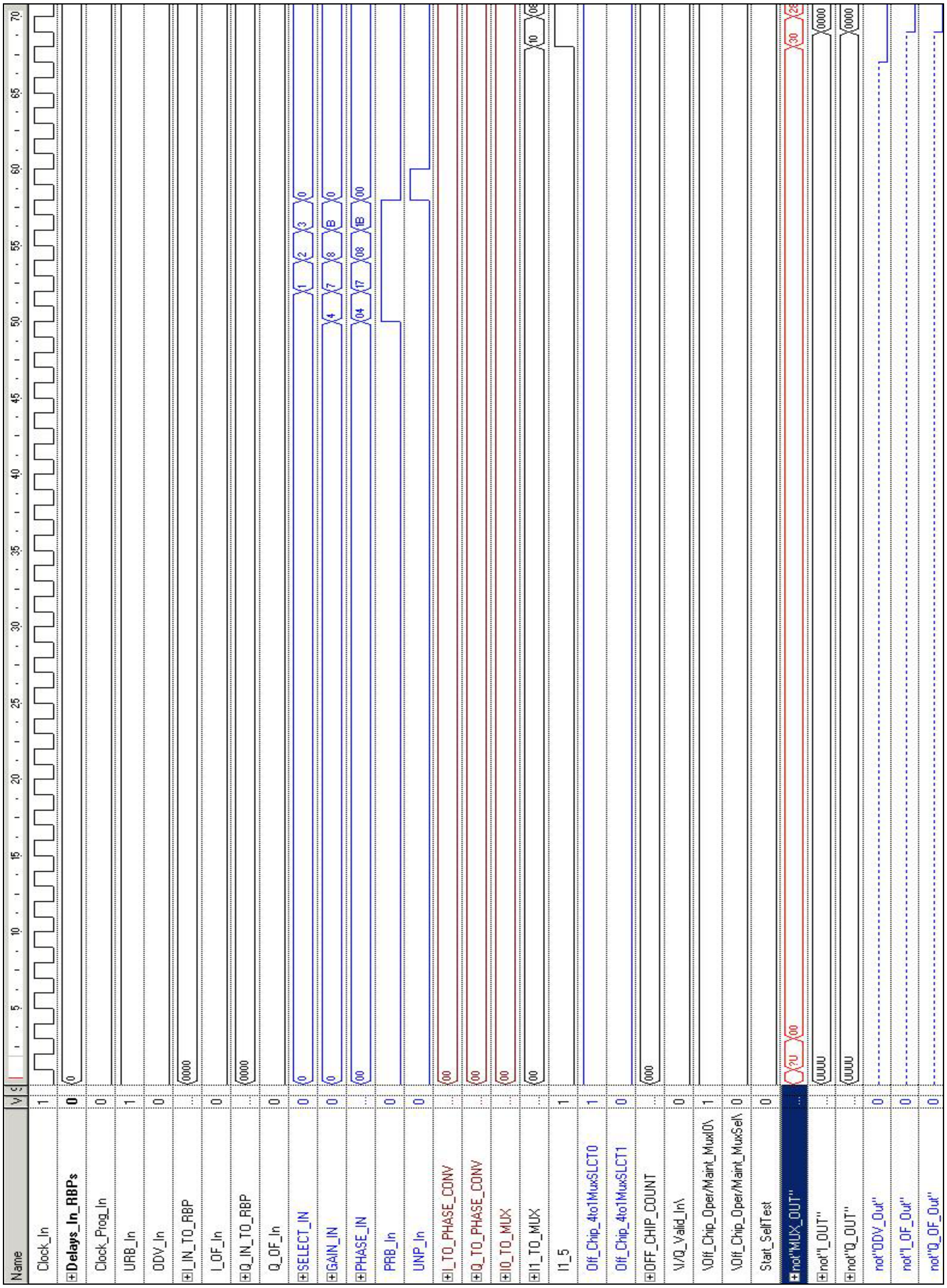


Figure 74. Simulation of the DIS – Path 2, Initialization

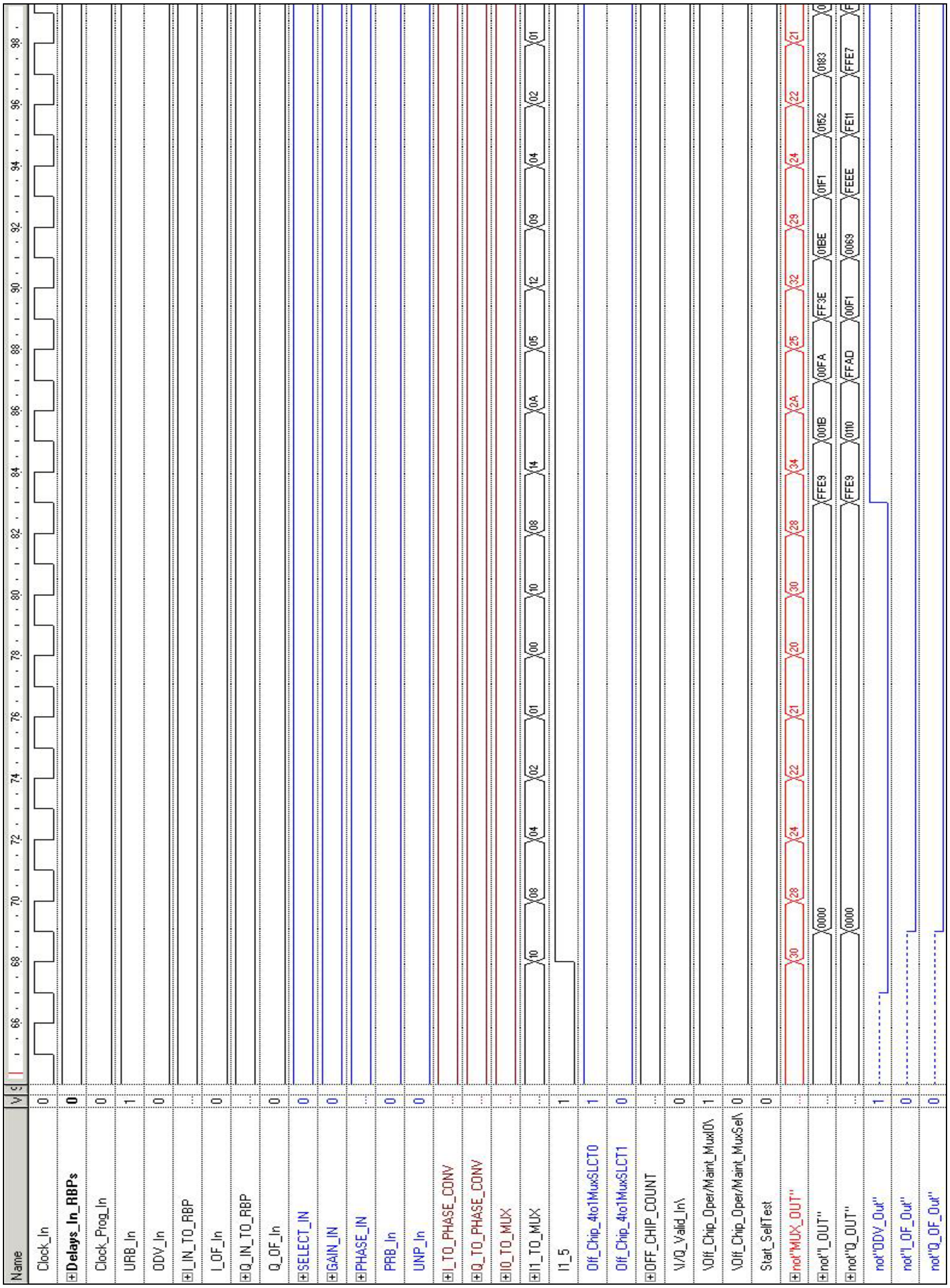


Figure 75. Simulation of the DIS – Path 2, Inputting Phase Samples

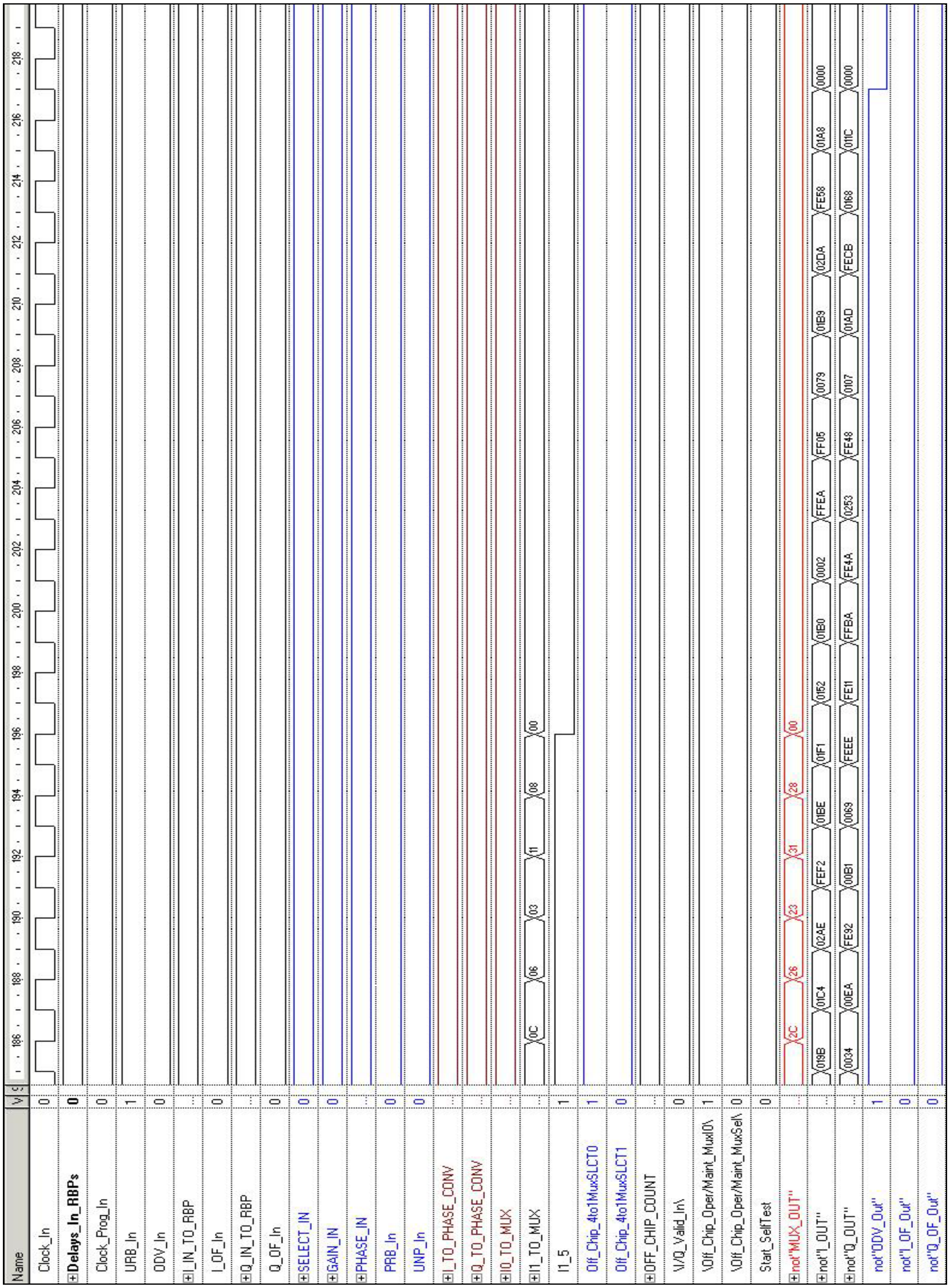


Figure 76. Simulation of the DIS – Path 2, Ending

RBP	0	1	2	3
Gain\(Hex)	04	07	08	0B
PInc(Hex)	04	17	08	1B

	Simulation Results					C++ Outputs				
I1 Phase Samples (Hex)	MUX Out (Hex)	I_Out (Hex)	Q_Out (Hex)	I_OF _Out	Q_OF _Out	MUX Out (Hex)	I_Out (Hex)	Q_Out (Hex)	I_OF _Out	Q_OF _Out
10	10	FFE9	FFE9	0	0	10	FFE9	FFE9	0	0
08	08	001B	0110	0	0	08	001B	0110	0	0
04	04	00FA	FFAD	0	0	04	00FA	FFAD	0	0
02	02	FF3E	00F1	0	0	02	FF3E	00F1	0	0
01	01	01BE	0069	0	0	01	01BE	0069	0	0
00	00	01F1	FEEE	0	0	00	01F1	FEEE	0	0
10	10	0152	FE11	0	0	10	0152	FE11	0	0
08	08	0183	FFE7	0	0	08	0183	FFE7	0	0
14	14	0216	FDC6	0	0	14	0216	FDC6	0	0
0A	0A	FDF8	0288	0	0	0A	FDF8	0288	0	0
05	05	02C8	0140	0	0	05	02C8	0140	0	0
12	12	FE78	FF79	0	0	12	FE78	FF79	0	0
09	09	009A	02D6	0	0	09	009A	02D6	0	0
04	04	0312	FFE4	0	0	04	0312	FFE4	0	0
02	02	FEB3	0058	0	0	02	FEB3	0058	0	0
01	01	017E	00B5	0	0	01	017E	00B5	0	0
10	10	01C4	FEC1	0	0	10	01C4	FEC1	0	0
08	08	01B6	0032	0	0	08	01B6	0032	0	0
14	14	0262	FE06	0	0	14	0262	FE06	0	0
0A	0A	FDF8	0288	0	0	0A	FDF8	0288	0	0
15	15	02D5	0101	0	0	15	02D5	0101	0	0
0A	0A	FCFF	010B	0	0	0A	FCFF	010B	0	0
05	05	0244	01D5	0	0	05	0244	01D5	0	0
12	12	FE70	FF15	0	0	12	FE70	FF15	0	0
09	09	009A	02D6	0	0	09	009A	02D6	0	0
04	04	0312	FFE4	0	0	04	0312	FFE4	0	0
02	02	FEB3	0058	0	0	02	FEB3	0058	0	0
11	11	015B	0080	0	0	11	015B	0080	0	0
08	08	01C4	00EA	0	0	08	01C4	00EA	0	0
04	04	02AE	FE92	0	0	04	02AE	FE92	0	0
12	12	FED9	0076	0	0	12	FED9	0076	0	0
09	09	012E	0254	0	0	09	012E	0254	0	0
14	14	030A	FF41	0	0	14	030A	FF41	0	0
1A	1A	FDA8	01D6	0	0	1A	FDA8	01D6	0	0
1D	1D	00BA	010F	0	0	1D	00BA	010F	0	0
0E	0E	FD74	FFBB	0	0	0E	FD74	FFBB	0	0
17	17	FFA6	FF46	0	0	17	FFA6	FF46	0	0
0B	0B	FEDD	FE31	0	0	0B	FEDD	FE31	0	0
15	15	00CA	0229	0	0	15	00CA	0229	0	0
0A	0A	FD25	FFD8	0	0	0A	FD25	FFD8	0	0

Table 23. Comparison of Simulation Results and C++ Outputs for Path 2

	Simulation Results					C++ Outputs				
I1 Phase Samples (Hex)	MUX Out (Hex)	I_Out (Hex)	Q_Out (Hex)	I_OF _Out	Q_OF _Out	MUX Out (Hex)	I_Out (Hex)	Q_Out (Hex)	I_OF _Out	Q_OF _Out
05	05	01EC	0201	0	0	05	01EC	0201	0	0
02	02	FE89	FF50	0	0	02	FE89	FF50	0	0
11	11	0107	00B6	0	0	11	0107	00B6	0	0
18	18	01F9	0121	0	0	18	01F9	0121	0	0
0C	0C	009A	FED7	0	0	0C	009A	FED7	0	0
06	06	FF9E	0213	0	0	06	FF9E	0213	0	0
03	03	FF05	FE48	0	0	03	FF05	FE48	0	0
11	11	0079	0107	0	0	11	0079	0107	0	0
08	08	01B9	01AD	0	0	08	01B9	01AD	0	0
04	04	02DA	FEEA	0	0	04	02DA	FEEA	0	0
-	-	FEF2	00B1	0	0	-	FEF2	00B1	0	0
-	-	019B	0034	0	0	-	019B	0034	0	0
-	-	01C4	00EA	0	0	-	01C4	00EA	0	0
-	-	02AE	FE92	0	0	-	02AE	FE92	0	0
-	-	FEF2	00B1	0	0	-	FEF2	00B1	0	0
-	-	01BE	0069	0	0	-	01BE	0069	0	0
-	-	01F1	FEEE	0	0	-	01F1	FEEE	0	0
-	-	0152	FE11	0	0	-	0152	FE11	0	0
-	-	01B0	FFBA	0	0	-	01B0	FFBA	0	0
-	-	0002	FE4A	0	0	-	0002	FE4A	0	0
-	-	FFEA	0253	0	0	-	FFEA	0253	0	0
-	-	FF05	FE48	0	0	-	FF05	FE48	0	0
-	-	0079	0107	0	0	-	0079	0107	0	0
-	-	01B9	01AD	0	0	-	01B9	01AD	0	0
-	-	02DA	FECB	0	0	-	02DA	FECB	0	0
-	-	FE58	0168	0	0	-	FE58	0168	0	0
-	-	01A8	011C	0	0	-	01A8	011C	0	0

Comparison of Simulation Results and C++ Outputs for Path 2, Continued

8. Simulation of Path 3 - Self Test Logic Circuit to RBP s

The simulation algorithm for Path 3, the flow from the self test logic circuit phase sample test vectors to the four RBP s connected serially, is given below.

- Set Clock_In = Stimulator → Clock → 2ns.
- Set Delay signals inside RBP s ='0'
- Set Clock_Prog_In = '0', URB_In='0'
- Set ODV_In, PRB_In and UNP_In ='0'

- Set I_In_0 through I_In_15, Q_In_0 through Q_In_15, I_OF_In, Q_OF_In, I0 through I7, Q0 through Q7, I1_0 through I1_5 and I0_0 through I0_5 =’0’

- Set Off_Chip_Count0 through Off_Chip_Count11 = To the desired number of test vectors to be generated, in this simulation it is 64.

- Set I/Q_Valid_In =’0’

- Set Start_SelfTest =’0’

- Set Off_Chip_4to1MuxSLCT0=’0’

- Set Off_Chip_4to1MuxSLCT1 =‘1’

- Set Off_Chip_Oper/Maint_MuxIO =’0’

- Set Off_Chip_Oper/Maint_MuxSel =’1’

- Clock the DIS for 23 times to clear the pipeline inside the RBP s.

- Set PRB_In =’1’

- Set Sel_In0 through Sel_In7 to the desired RBP number; set Gain_In_0 through Gain_In_3 and Phase_In_0 through Phase_In_4 to the proper coefficient values. Clock the DIS once. Repeat for every RBP to be programmed.

- Set PRB = ‘0’, UNP=’1’, clock the DIS once

- Set Start_SelfTest =’1’

- Clock the DIS for as many as the number of the test vectors, until the I_Out and Q_Out values “freeze”

- Observe MUX_Out0 through MUX_Out5 to verify the inputs are steered into the RBP s from the 6-bit 4-to-1 multiplexer.

- Observe ODV_Out, I_Out_0 through I_Out_15, Q_Out_0 through Q_Out_15, I_OF_Out and Q_OF_Out.

- Compare the results with the C++ simulation outputs.

Table 24 shows the RBP programming coefficients, phase sample input values to the first data path, signal values probed at the output of 6-bit 4-to-1 multiplexer and outputs of the DIS.

The waveform editor used to simulate the DIS for the first data path is given in Figures 77 and 78, showing initialization and the end of simulation, respectively.

RBP	0	1	2	3
Gain\(Hex)	04	07	08	0B
PInc(Hex)	04	17	08	1B

	Simulation Results					C++ Outputs				
Self Test Outputs (Hex)	MUX Out (Hex)	I_Out (Hex)	Q_Out (Hex)	I_OF _Out	Q_OF _Out	MUX Out (Hex)	I_Out (Hex)	Q_Out (Hex)	I_OF _Out	Q_OF _Out
10	10	FFE9	FFE9	0	0	10	FFE9	FFE9	0	0
08	08	001B	0110	0	0	08	001B	0110	0	0
04	04	00FA	FFAD	0	0	04	00FA	FFAD	0	0
02	02	FF3E	00F1	0	0	02	FF3E	00F1	0	0
01	01	01BE	0069	0	0	01	01BE	0069	0	0
00	00	01F1	FEEE	0	0	00	01F1	FEEE	0	0
10	10	0152	FE11	0	0	10	0152	FE11	0	0
08	08	0183	FFE7	0	0	08	0183	FFE7	0	0
14	14	0216	FDC6	0	0	14	0216	FDC6	0	0
0A	0A	FDF8	0288	0	0	0A	FDF8	0288	0	0
05	05	02C8	0140	0	0	05	02C8	0140	0	0
12	12	FE78	FF79	0	0	12	FE78	FF79	0	0
09	09	009A	02D6	0	0	09	009A	02D6	0	0
04	04	0312	FFE4	0	0	04	0312	FFE4	0	0
02	02	FEB3	0058	0	0	02	FEB3	0058	0	0
01	01	017E	00B5	0	0	01	017E	00B5	0	0
10	10	01C4	FEC1	0	0	10	01C4	FEC1	0	0
08	08	01B6	0032	0	0	08	01B6	0032	0	0
14	14	0262	FE06	0	0	14	0262	FE06	0	0
0A	0A	FDF8	0288	0	0	0A	FDF8	0288	0	0
15	15	02D5	0101	0	0	15	02D5	0101	0	0
0A	0A	FCFF	010B	0	0	0A	FCFF	010B	0	0
05	05	0244	01D5	0	0	05	0244	01D5	0	0
12	12	FE70	FF15	0	0	12	FE70	FF15	0	0
09	09	009A	02D6	0	0	09	009A	02D6	0	0
04	04	0312	FFE4	0	0	04	0312	FFE4	0	0
02	02	FEB3	0058	0	0	02	FEB3	0058	0	0

Table 24. Comparison of Simulation Results and C++ Outputs for Path 3

	Simulation Results					C++ Outputs				
Self Test Outputs (Hex)	MUX Out (Hex)	I_Out (Hex)	Q_Out (Hex)	I_OF _Out	Q_OF _Out	MUX Out (Hex)	I_Out (Hex)	Q_Out (Hex)	I_OF _Out	Q_OF _Out
11	11	015B	0080	0	0	11	015B	0080	0	0
08	08	01C4	00EA	0	0	08	01C4	00EA	0	0
04	04	02AE	FE92	0	0	04	02AE	FE92	0	0
12	12	FED9	0076	0	0	12	FED9	0076	0	0
09	09	012E	0254	0	0	09	012E	0254	0	0
14	14	030A	FF41	0	0	14	030A	FF41	0	0
1A	1A	FDA8	01D6	0	0	1A	FDA8	01D6	0	0
1D	1D	00BA	010F	0	0	1D	00BA	010F	0	0
0E	0E	FD74	FFBB	0	0	0E	FD74	FFBB	0	0
17	17	FFA6	FF46	0	0	17	FFA6	FF46	0	0
0B	0B	FEDD	FE31	0	0	0B	FEDD	FE31	0	0
15	15	00CA	0229	0	0	15	00CA	0229	0	0
0A	0A	FD25	FFD8	0	0	0A	FD25	FFD8	0	0
05	05	01EC	0201	0	0	05	01EC	0201	0	0
02	02	FE89	FF50	0	0	02	FE89	FF50	0	0
11	11	0107	00B6	0	0	11	0107	00B6	0	0
18	18	01F9	0121	0	0	18	01F9	0121	0	0
0C	0C	009A	FED7	0	0	0C	009A	FED7	0	0
06	06	FF9E	0213	0	0	06	FF9E	0213	0	0
03	03	FF05	FE48	0	0	03	FF05	FE48	0	0
11	11	0079	0107	0	0	11	0079	0107	0	0
08	08	01B9	01AD	0	0	08	01B9	01AD	0	0
04	04	02DA	FEEA	0	0	04	02DA	FEEA	0	0
-	-	FEF2	00B1	0	0	-	FEF2	00B1	0	0
-	-	019B	0034	0	0	-	019B	0034	0	0
-	-	01C4	00EA	0	0	-	01C4	00EA	0	0
-	-	02AE	FE92	0	0	-	02AE	FE92	0	0
-	-	FEF2	00B1	0	0	-	FEF2	00B1	0	0
-	-	01BE	0069	0	0	-	01BE	0069	0	0
-	-	01F1	FEEE	0	0	-	01F1	FEEE	0	0
-	-	0152	FE11	0	0	-	0152	FE11	0	0
-	-	01B0	FFBA	0	0	-	01B0	FFBA	0	0
-	-	0002	FE4A	0	0	-	0002	FE4A	0	0
-	-	FFEA	0253	0	0	-	FFEA	0253	0	0

Comparison of Simulation Results and C++ Outputs for Path 3, Continued

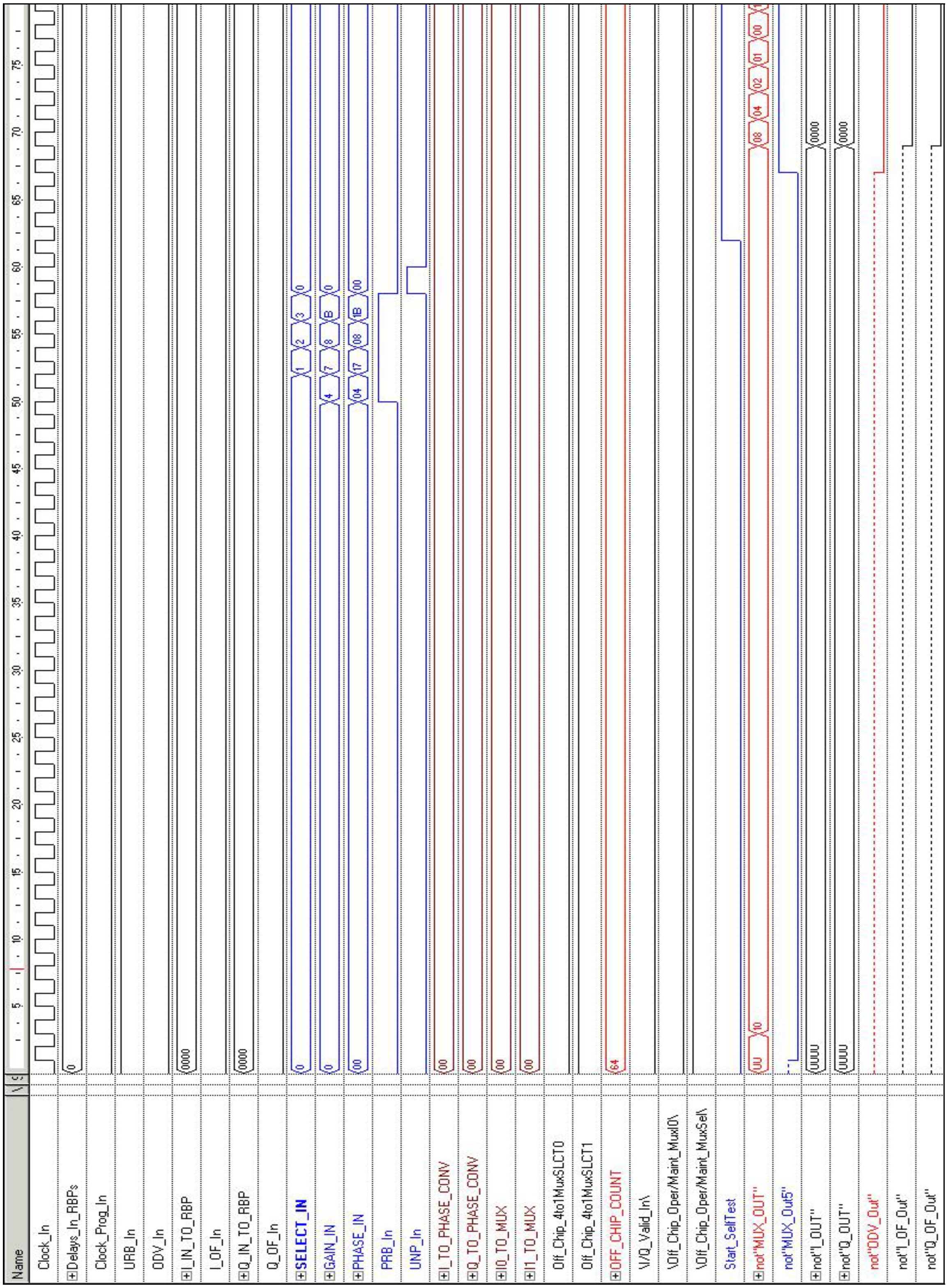


Figure 77. Simulation of the DIS – Path 3, Initialization

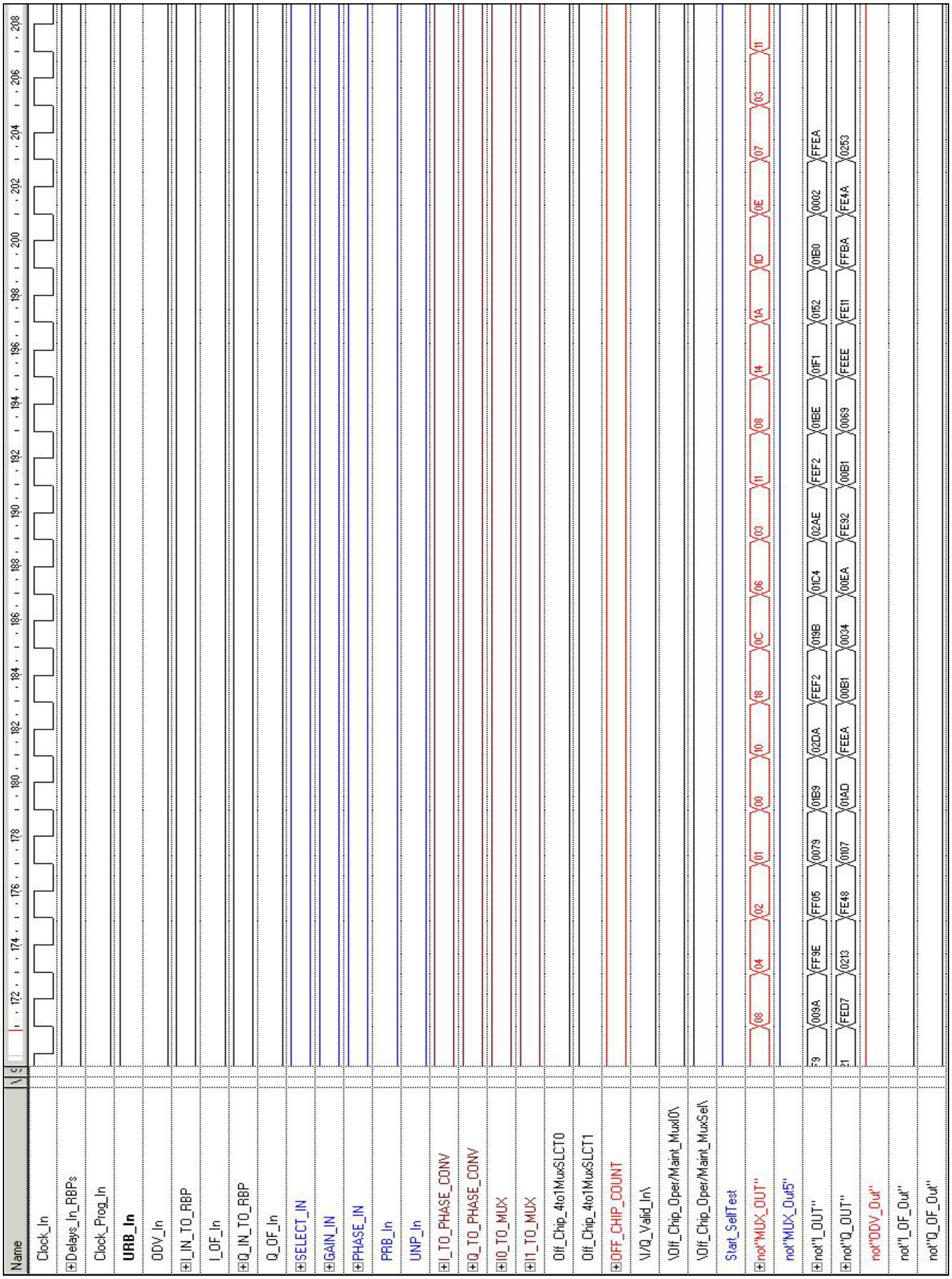


Figure 78. Simulation of the DIS – Path 3, Ending

9. Simulation of Path 4 - Phase Extraction Circuit to RBP s

The simulation algorithm for Path 4, the flow from the phase extraction circuit to the four RBP s connected serially, is given below.

- Set Clock_In = Stimulator → Clock → 2ns.
- Set Delay signals inside RBP s =’0’
- Set Clock_Prog_In = ‘0’, URB_In=’0’
- Set ODV_In, PRB_In and UNP_In =’0’
- Set I_In_0 through I_In_15, Q_In_0 through Q_In_15, I_OF_In, Q_OF_In, I0 through I7, Q0 through Q7, I1_0 through I1_5 and I0_0 through I0_5 =’0’
- Set Off_Chip_Count0 through Off_Chip_Count11 = ‘0’
- Set I/Q_Valid_In =’0’
- Set Start_SelfTest =’0’
- Set Off_Chip_4to1MuxSLCT0=’1’
- Set Off_Chip_4to1MuxSLCT1 =‘1’
- Set Off_Chip_Oper/Maint_MuxIO =’1’
- Set Off_Chip_Oper/Maint_MuxSel =’0’
- Clock the DIS for 23 times to clear the pipeline inside the RBP s.
- Set PRB_In =’1’
- Set Sel_In0 through Sel_In7 to the desired RBP number; set Gain_In_0 through Gain_In_3 and Phase_In_0 through Phase_In_4 to the proper coefficient values. Clock the DIS once. Repeat for every RBP to be programmed.
- Set PRB = ‘0’, UNP=’1’, clock the DIS once
- Clock the DIS until ODV_Out becomes “low”
- Set I/Q_Valid_In =’1’

- Set the I/Q sample value by modifying I0 through I7 and Q0 through Q7. Clock the DIS once. Repeat for every phase sample value.

- Observe MUX_Out0 through MUX_Out5 to verify the inputs are steered into the RBP s from the 6-bit 4-to-1 multiplexer.

- Observe ODV_Out, I_Out_0 through I_Out_15, Q_Out_0 through Q_Out_15, I_OF_Out and Q_OF_Out.

- Compare the results with the C++ simulation outputs.

Table 25 shows the RBP programming coefficients, phase sample input values to the fourth data path, signal values probed at the output of 6-bit 4-to-1 multiplexer and the outputs of the DIS.

The waveform editor used to simulate the DIS for the first data path is given in Figures 79 and 80, showing initialization and the end of simulation, respectively.

RBP	**0**	**1**	**2**	**3**
Gain\(Hex)	04	07	08	0B
PInc(Hex)	04	17	08	1B

		Simulation Results				**C++ Outputs**				
I values to Phase Extractor	**Q values to Phase Extractor**	**I_Out (Hex)**	**Q_Out (Hex)**	**I_OF _Out**	**Q_OF _Out**	**MUX Out (Hex)**	**I_Out (Hex)**	**Q_Out (Hex)**	**I_OF _Out**	**Q_OF _Out**
80	FC	FFE9	FFE9	0	0	10	FFE9	FFE9	0	0
FF	1F	001B	0110	0	0	08	001B	0110	0	0
03	03	00FA	FFAD	0	0	04	00FA	FFAD	0	0
06	02	FF3E	00F1	0	0	02	FF3E	00F1	0	0
05	01	01BE	0069	0	0	01	01BE	0069	0	0
01	00	01F1	FEEE	0	0	00	01F1	FEEE	0	0
06	FC	0152	FE11	0	0	10	0152	FE11	0	0
80	1F	0183	FFE7	0	0	08	0183	FFE7	0	0
FF	FF	0216	FDC6	0	0	14	0216	FDC6	0	0
FF	05	FDF8	0288	0	0	0A	FDF8	0288	0	0
FE	03	02C8	0140	0	0	05	02C8	0140	0	0
02	CC	FE78	FF79	0	0	12	FE78	FF79	0	0
80	0D	009A	02D6	0	0	09	009A	02D6	0	0
FE	03	0312	FFE4	0	0	04	0312	FFE4	0	0
03	02	FEB3	0058	0	0	02	FEB3	0058	0	0
06	01	017E	00B5	0	0	01	017E	00B5	0	0
05	FC	01C4	FEC1	0	0	10	01C4	FEC1	0	0

Table 25. Comparison of Simulation Results and C++ Outputs for Path 4

		Simulation Results					**C++ Outputs**			
I values to Phase Extractor	**Q values to Phase Extractor**	**I_Out (Hex)**	**Q_Out (Hex)**	**I_OF _Out**	**Q_OF _Out**	**MUX Out (Hex)**	**I_Out (Hex)**	**Q_Out (Hex)**	**I_OF _Out**	**Q_OF _Out**
80	1F	01B6	0032	0	0	08	01B6	0032	0	0
FF	FF	0262	FE06	0	0	14	0262	FE06	0	0
FE	05	FDF8	0288	0	0	0A	FDF8	0288	0	0
FE	FD	02D5	0101	0	0	15	02D5	0101	0	0
FE	05	FCFF	010B	0	0	0A	FCFF	010B	0	0
02	03	0244	01D5	0	0	05	0244	01D5	0	0
80	CC	FE70	FF15	0	0	12	FE70	FF15	0	0
FE	0D	009A	02D6	0	0	09	009A	02D6	0	0
03	03	0312	FFE4	0	0	04	0312	FFE4	0	0
06	02	FEB3	0058	0	0	02	FEB3	0058	0	0
FC	FF	015B	0080	0	0	11	015B	0080	0	0
FF	1F	01C4	00EA	0	0	08	01C4	00EA	0	0
03	03	02AE	FE92	0	0	04	02AE	FE92	0	0
80	CC	FED9	0076	0	0	12	FED9	0076	0	0
FE	0D	012E	0254	0	0	09	012E	0254	0	0
FF	FF	030A	FF41	0	0	14	030A	FF41	0	0
06	F3	FDA8	01D6	0	0	1A	FDA8	01D6	0	0
03	FE	00BA	010F	0	0	1D	00BA	010F	0	0
80	33	FD74	FFBB	0	0	0E	FD74	FFBB	0	0
FE	F2	FFA6	FF46	0	0	17	FFA6	FF46	0	0
FE	03	FEDD	FE31	0	0	0B	FEDD	FE31	0	0
FE	FD	00CA	0229	0	0	15	00CA	0229	0	0
FE	05	FD25	FFD8	0	0	0A	FD25	FFD8	0	0
02	03	01EC	0201	0	0	05	01EC	0201	0	0
06	02	FE89	FF50	0	0	02	FE89	FF50	0	0
FC	FF	0107	00B6	0	0	11	0107	00B6	0	0
02	DF	01F9	0121	0	0	18	01F9	0121	0	0
FF	01	009A	FED7	0	0	0C	009A	FED7	0	0
0B	1D	FF9E	0213	0	0	06	FF9E	0213	0	0
03	02	FF05	FE48	0	0	03	FF05	FE48	0	0
FC	FF	0079	0107	0	0	11	0079	0107	0	0
FF	1F	01B9	01AD	0	0	08	01B9	01AD	0	0
03	03	02DA	FEEA	0	0	04	02DA	FEEA	0	0
-	-	FEE6	0094	0	0	-	FEE6	0094	0	0
-	-	017B	0149	0	0	-	017B	0149	0	0
-	-	01F4	FF9C	0	0	-	01F4	FF9C	0	0

Comparison of Simulation Results and C++ Outputs for Path 4, Continued

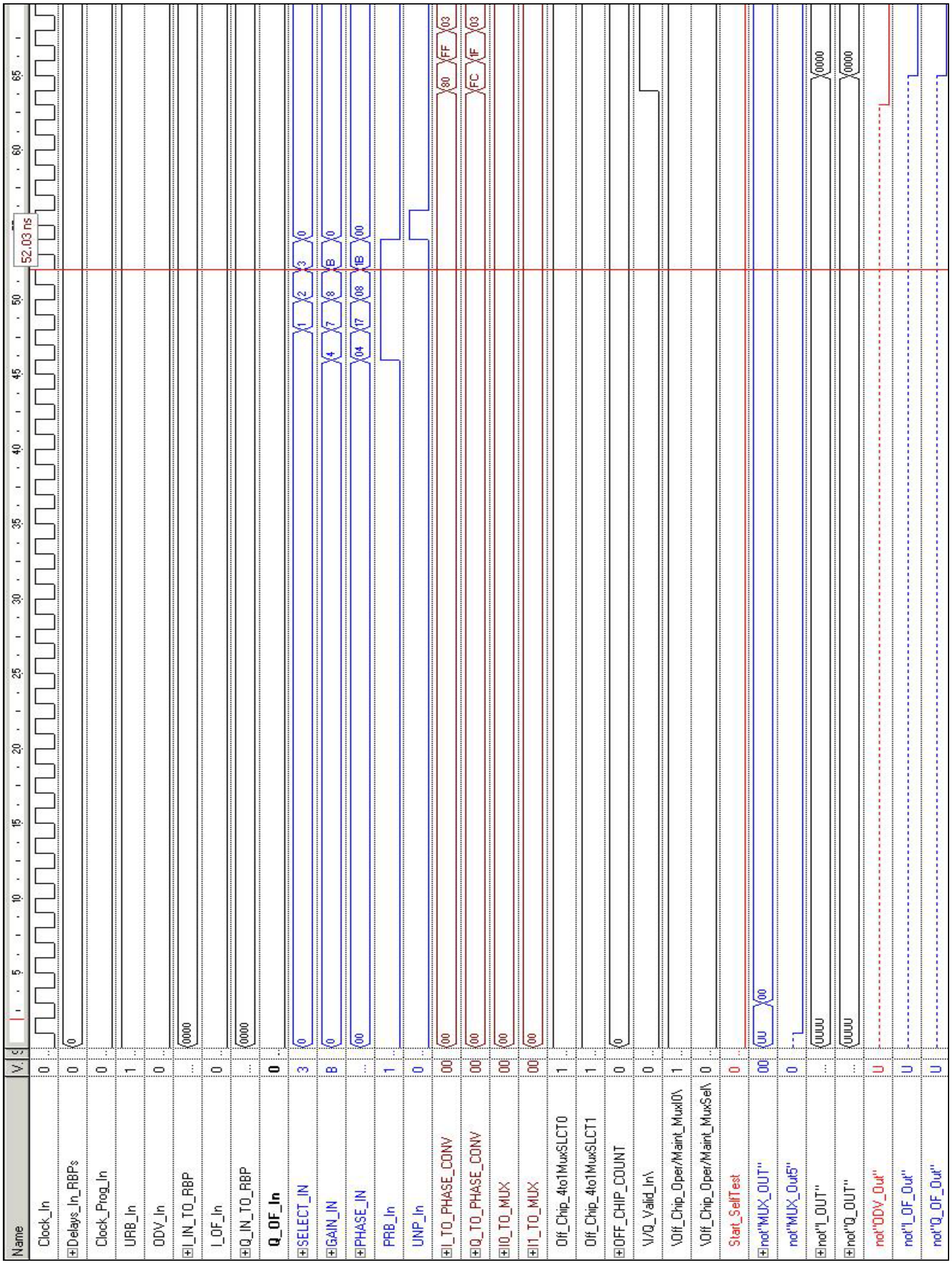


Figure 79. Simulation of the DIS – Path 4, Initialization

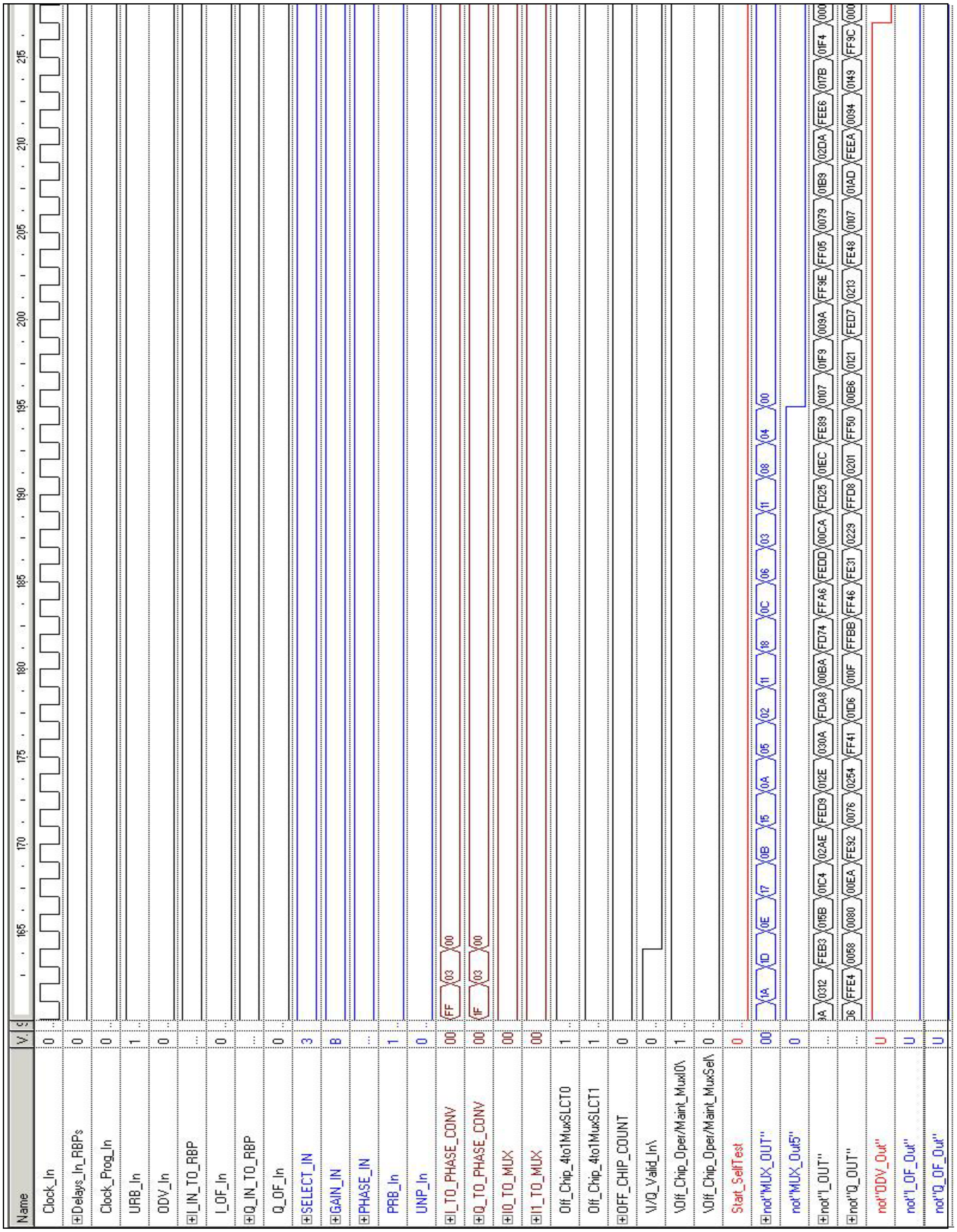


Figure 80. Simulation of the DIS – Path 4, Ending

Simulations performed in this chapter shows that the DIS operates correctly. Main functional blocks and data paths were tested and verified to operate. Chapter VI provides conclusions about the results obtained during this thesis and summarizes the future research opportunities in the DIS project.

VI. CONCLUSION

A. RESULTS AND CONCLUSION

The main purpose of the research for this thesis was to model, test and verify a Digital Image Synthesizer (DIS) implemented on a full-custom Application Specific Integrated Circuit (ASIC) to counter Inverse Synthetic Aperture Radars (ISARs).

Due to difficulties in other methods of verification for large electronic systems, testing and verification of the system was performed in a hardware description language environment, VHDL. The VHDL code, since it is automatically generated, was not optimum in size. Some problems with the simulation software were encountered. Although the research group tried to address the software defects, even the vendor of the simulation tool was unable to fix the "bugs" in time. This fact hindered testing of the DIS with 512 Range Bin Processors (RBPs). However, since the RBPs are identical, testing and verification of the DIS with 4 RBP s was found a safe method to implement.

Testing and verification efforts were conducted in parallel with the design process. It provided almost instant feedback to the design team and saved time. Furthermore, the testing algorithms for different components were made easier with the help of the design team.

VHDL simulations for low-level components were tested and verified for proper operation. This provided a starting point for larger components and allowed a straightforward testing and verification plan.

Larger components and basic data flow paths in the DIS were confirmed to operate correctly. Some components were defined in their behavioral descriptions.

Finally, functionality of the DIS chip was tested and verified.

B. FUTURE WORK

The 512 RBP s and the control circuitry can be tested if the simulation software is upgraded and fixed to accommodate larger size circuits.

The DIS chip is to be fabricated in the summer of 2003. More functional testing and timing analysis should be conducted on the actual hardware implementation.

The chip should also be tested with the other hardware components such as the Digital Radio Frequency Memory (DRFM).

APPENDIX A – TEST VECTORS

This appendix contains the Phase Sample Value Vectors created by the Self Test Circuit. It can create up to 4095 pseudo-random test sequence to test the DIS. Chapter II has more information about the Self Test Logic, while Chapter V presents the methodology followed to use the Self Test Circuitry with the control inputs.

The table on the subsequent pages gives a complete list of generated test vectors.

NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value
0000	0x10	0045	0x06	0090	0x17	0135	0x18	0180	0x0B	0225	0x05	0270	0x15
0001	0x08	0046	0x03	0091	0x0B	0136	0x1C	0181	0x05	0226	0x02	0271	0x0A
0002	0x04	0047	0x11	0092	0x05	0137	0x0E	0182	0x02	0227	0x11	0272	0x15
0003	0x02	0048	0x08	0093	0x02	0138	0x07	0183	0x11	0228	0x08	0273	0x0A
0004	0x01	0049	0x04	0094	0x01	0139	0x13	0184	0x08	0229	0x04	0274	0x15
0005	0x00	0050	0x02	0095	0x10	0140	0x19	0185	0x14	0230	0x02	0275	0x0A
0006	0x10	0051	0x11	0096	0x18	0141	0x1C	0186	0x0A	0231	0x11	0276	0x05
0007	0x08	0052	0x08	0097	0x1C	0142	0x0E	0187	0x15	0232	0x18	0277	0x02
0008	0x14	0053	0x04	0098	0x1E	0143	0x17	0188	0x0A	0233	0x0C	0278	0x11
0009	0x0A	0054	0x02	0099	0x0F	0144	0x1B	0189	0x05	0234	0x16	0279	0x18
0010	0x05	0055	0x01	0100	0x17	0145	0x0D	0190	0x12	0235	0x0B	0280	0x1C
0011	0x12	0056	0x00	0101	0x0B	0146	0x16	0191	0x19	0236	0x05	0281	0x1E
0012	0x09	0057	0x10	0102	0x05	0147	0x0B	0192	0x0C	0237	0x12	0282	0x0F
0013	0x04	0058	0x18	0103	0x02	0148	0x05	0193	0x06	0238	0x09	0283	0x17
0014	0x02	0059	0x0C	0104	0x01	0149	0x02	0194	0x13	0239	0x04	0284	0x0B
0015	0x01	0060	0x06	0105	0x10	0150	0x01	0195	0x09	0240	0x02	0285	0x05
0016	0x10	0061	0x03	0106	0x18	0151	0x10	0196	0x04	0241	0x01	0286	0x12
0017	0x08	0062	0x11	0107	0x0C	0152	0x08	0197	0x02	0242	0x00	0287	0x09
0018	0x14	0063	0x08	0108	0x16	0153	0x14	0198	0x01	0243	0x10	0288	0x14
0019	0x0A	0064	0x14	0109	0x0B	0154	0x0A	0199	0x00	0244	0x18	0289	0x1A
0020	0x15	0065	0x1A	0110	0x15	0155	0x05	0200	0x10	0245	0x0C	0290	0x0D
0021	0x0A	0066	0x1D	0111	0x0A	0156	0x12	0201	0x18	0246	0x16	0291	0x16
0022	0x05	0067	0x0E	0112	0x05	0157	0x09	0202	0x1C	0247	0x0B	0292	0x1B
0023	0x12	0068	0x07	0113	0x12	0158	0x14	0203	0x1E	0248	0x15	0293	0x1D
0024	0x09	0069	0x03	0114	0x09	0159	0x0A	0204	0x0F	0249	0x0A	0294	0x1E
0025	0x04	0070	0x11	0115	0x04	0160	0x05	0205	0x17	0250	0x15	0295	0x0F
0026	0x02	0071	0x18	0116	0x12	0161	0x12	0206	0x0B	0251	0x1A	0296	0x17
0027	0x11	0072	0x0C	0117	0x09	0162	0x09	0207	0x15	0252	0x0D	0297	0x1B
0028	0x08	0073	0x06	0118	0x04	0163	0x14	0208	0x0A	0253	0x06	0298	0x0D
0029	0x04	0074	0x03	0119	0x02	0164	0x1A	0209	0x05	0254	0x13	0299	0x16
0030	0x12	0075	0x11	0120	0x11	0165	0x1D	0210	0x12	0255	0x09	0300	0x0B
0031	0x09	0076	0x18	0121	0x08	0166	0x1E	0211	0x19	0256	0x14	0301	0x05
0032	0x14	0077	0x1C	0122	0x04	0167	0x0F	0212	0x0C	0257	0x0A	0302	0x12
0033	0x1A	0078	0x0E	0123	0x02	0168	0x17	0213	0x06	0258	0x05	0303	0x19
0034	0x1D	0079	0x17	0124	0x01	0169	0x1B	0214	0x03	0259	0x02	0304	0x1C
0035	0x0E	0080	0x0B	0125	0x10	0170	0x0D	0215	0x01	0260	0x01	0305	0x0E
0036	0x17	0081	0x05	0126	0x18	0171	0x06	0216	0x00	0261	0x10	0306	0x17
0037	0x0B	0082	0x12	0127	0x1C	0172	0x13	0217	0x00	0262	0x08	0307	0x0B
0038	0x15	0083	0x19	0128	0x0E	0173	0x09	0218	0x00	0263	0x14	0308	0x15
0039	0x0A	0084	0x1C	0129	0x07	0174	0x14	0219	0x00	0264	0x1A	0309	0x0A
0040	0x05	0085	0x0E	0130	0x03	0175	0x1A	0220	0x00	0265	0x1D	0310	0x15
0041	0x02	0086	0x17	0131	0x01	0176	0x1D	0221	0x10	0266	0x1E	0311	0x0A
0042	0x11	0087	0x1B	0132	0x00	0177	0x1E	0222	0x08	0267	0x0F	0312	0x05
0043	0x18	0088	0x1D	0133	0x00	0178	0x0F	0223	0x14	0268	0x17	0313	0x12
0044	0x0C	0089	0x0E	0134	0x10	0179	0x17	0224	0x0A	0269	0x0B	0314	0x09

NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value
0315	0x04	0360	0x1D	0405	0x0A	0450	0x15	0495	0x00	0540	0x1F	0585	0x04
0316	0x12	0361	0x1E	0406	0x05	0451	0x0A	0496	0x00	0541	0x1F	0586	0x02
0317	0x19	0362	0x0F	0407	0x02	0452	0x15	0497	0x10	0542	0x0F	0587	0x11
0318	0x0C	0363	0x17	0408	0x01	0453	0x1A	0498	0x08	0543	0x17	0588	0x18
0319	0x06	0364	0x0B	0409	0x00	0454	0x1D	0499	0x14	0544	0x1B	0589	0x1C
0320	0x13	0365	0x15	0410	0x10	0455	0x0E	0500	0x1A	0545	0x0D	0590	0x1E
0321	0x09	0366	0x1A	0411	0x18	0456	0x07	0501	0x0D	0546	0x16	0591	0x0F
0322	0x04	0367	0x0D	0412	0x1C	0457	0x13	0502	0x06	0547	0x1B	0592	0x17
0323	0x12	0368	0x06	0413	0x1E	0458	0x19	0503	0x13	0548	0x1D	0593	0x1B
0324	0x09	0369	0x13	0414	0x1F	0459	0x0C	0504	0x09	0549	0x0E	0594	0x0D
0325	0x04	0370	0x09	0415	0x1F	0460	0x16	0505	0x04	0550	0x17	0595	0x16
0326	0x12	0371	0x04	0416	0x0F	0461	0x0B	0506	0x12	0551	0x1B	0596	0x0B
0327	0x19	0372	0x12	0417	0x17	0462	0x05	0507	0x19	0552	0x0D	0597	0x15
0328	0x1C	0373	0x09	0418	0x0B	0463	0x02	0508	0x0C	0553	0x06	0598	0x1A
0329	0x0E	0374	0x04	0419	0x05	0464	0x11	0509	0x06	0554	0x13	0599	0x1D
0330	0x07	0375	0x02	0420	0x02	0465	0x18	0510	0x13	0555	0x19	0600	0x1E
0331	0x03	0376	0x11	0421	0x11	0466	0x1C	0511	0x19	0556	0x1C	0601	0x0F
0332	0x01	0377	0x18	0422	0x18	0467	0x0E	0512	0x0C	0557	0x0E	0602	0x17
0333	0x10	0378	0x0C	0423	0x0C	0468	0x07	0513	0x16	0558	0x17	0603	0x1B
0334	0x18	0379	0x06	0424	0x06	0469	0x03	0514	0x0B	0559	0x1B	0604	0x1D
0335	0x0C	0380	0x03	0425	0x13	0470	0x01	0515	0x05	0560	0x0D	0605	0x1E
0336	0x16	0381	0x11	0426	0x09	0471	0x00	0516	0x12	0561	0x06	0606	0x1F
0337	0x1B	0382	0x18	0427	0x04	0472	0x10	0517	0x09	0562	0x03	0607	0x0F
0338	0x0D	0383	0x0C	0428	0x02	0473	0x18	0518	0x14	0563	0x01	0608	0x17
0339	0x16	0384	0x06	0429	0x01	0474	0x1C	0519	0x1A	0564	0x00	0609	0x1B
0340	0x0B	0385	0x13	0430	0x10	0475	0x1E	0520	0x0D	0565	0x00	0610	0x0D
0341	0x15	0386	0x09	0431	0x18	0476	0x0F	0521	0x06	0566	0x10	0611	0x06
0342	0x0A	0387	0x04	0432	0x1C	0477	0x07	0522	0x13	0567	0x18	0612	0x13
0343	0x15	0388	0x12	0433	0x1E	0478	0x03	0523	0x19	0568	0x1C	0613	0x09
0344	0x0A	0389	0x09	0434	0x1F	0479	0x11	0524	0x1C	0569	0x1E	0614	0x04
0345	0x05	0390	0x14	0435	0x0F	0480	0x08	0525	0x0E	0570	0x0F	0615	0x12
0346	0x12	0391	0x1A	0436	0x07	0481	0x04	0526	0x17	0571	0x17	0616	0x19
0347	0x19	0392	0x1D	0437	0x03	0482	0x12	0527	0x0B	0572	0x1B	0617	0x1C
0348	0x0C	0393	0x1E	0438	0x01	0483	0x09	0528	0x05	0573	0x1D	0618	0x0E
0349	0x16	0394	0x0F	0439	0x00	0484	0x04	0529	0x02	0574	0x0E	0619	0x17
0350	0x1B	0395	0x07	0440	0x00	0485	0x12	0530	0x01	0575	0x07	0620	0x1B
0351	0x0D	0396	0x13	0441	0x00	0486	0x09	0531	0x00	0576	0x13	0621	0x0D
0352	0x06	0397	0x09	0442	0x10	0487	0x04	0532	0x00	0577	0x19	0622	0x16
0353	0x03	0398	0x04	0443	0x08	0488	0x12	0533	0x10	0578	0x1C	0623	0x1B
0354	0x01	0399	0x12	0444	0x04	0489	0x09	0534	0x18	0579	0x0E	0624	0x0D
0355	0x10	0400	0x09	0445	0x02	0490	0x04	0535	0x0C	0580	0x17	0625	0x06
0356	0x18	0401	0x04	0446	0x11	0491	0x02	0536	0x16	0581	0x0B	0626	0x03
0357	0x0C	0402	0x12	0447	0x08	0492	0x01	0537	0x1B	0582	0x05	0627	0x11
0358	0x16	0403	0x09	0448	0x14	0493	0x00	0538	0x1D	0583	0x12	0628	0x08
0359	0x1B	0404	0x14	0449	0x0A	0494	0x00	0539	0x1E	0584	0x09	0629	0x04

NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value
0630	0x02	0675	0x08	0720	0x04	0765	0x14	0810	0x0D	0855	0x1E	0900	0x04
0631	0x11	0676	0x14	0721	0x12	0766	0x1A	0811	0x06	0856	0x0F	0901	0x02
0632	0x18	0677	0x1A	0722	0x19	0767	0x0D	0812	0x13	0857	0x07	0902	0x11
0633	0x0C	0678	0x1D	0723	0x1C	0768	0x16	0813	0x09	0858	0x13	0903	0x18
0634	0x06	0679	0x1E	0724	0x1E	0769	0x0B	0814	0x14	0859	0x09	0904	0x1C
0635	0x03	0680	0x1F	0725	0x0F	0770	0x15	0815	0x1A	0860	0x14	0905	0x0E
0636	0x01	0681	0x0F	0726	0x17	0771	0x0A	0816	0x0D	0861	0x1A	0906	0x07
0637	0x10	0682	0x07	0727	0x1B	0772	0x15	0817	0x06	0862	0x0D	0907	0x13
0638	0x08	0683	0x13	0728	0x1D	0773	0x1A	0818	0x03	0863	0x06	0908	0x19
0639	0x04	0684	0x09	0729	0x1E	0774	0x1D	0819	0x11	0864	0x13	0909	0x0C
0640	0x12	0685	0x14	0730	0x1F	0775	0x1E	0820	0x08	0865	0x09	0910	0x16
0641	0x09	0686	0x1A	0731	0x1F	0776	0x1F	0821	0x04	0866	0x04	0911	0x1B
0642	0x14	0687	0x0D	0732	0x1F	0777	0x0F	0822	0x12	0867	0x02	0912	0x1D
0643	0x1A	0688	0x16	0733	0x1F	0778	0x17	0823	0x09	0868	0x11	0913	0x0E
0644	0x1D	0689	0x1B	0734	0x0F	0779	0x0B	0824	0x14	0869	0x08	0914	0x17
0645	0x1E	0690	0x0D	0735	0x07	0780	0x15	0825	0x0A	0870	0x04	0915	0x1B
0646	0x1F	0691	0x06	0736	0x13	0781	0x1A	0826	0x05	0871	0x12	0916	0x1D
0647	0x0F	0692	0x03	0737	0x19	0782	0x1D	0827	0x02	0872	0x19	0917	0x1E
0648	0x17	0693	0x11	0738	0x0C	0783	0x0E	0828	0x11	0873	0x1C	0918	0x1F
0649	0x0B	0694	0x18	0739	0x06	0784	0x07	0829	0x08	0874	0x1E	0919	0x1F
0650	0x05	0695	0x0C	0740	0x13	0785	0x03	0830	0x14	0875	0x1F	0920	0x1F
0651	0x12	0696	0x06	0741	0x19	0786	0x01	0831	0x1A	0876	0x0F	0921	0x0F
0652	0x19	0697	0x13	0742	0x1C	0787	0x10	0832	0x1D	0877	0x17	0922	0x07
0653	0x0C	0698	0x19	0743	0x1E	0788	0x18	0833	0x1E	0878	0x1B	0923	0x03
0654	0x06	0699	0x0C	0744	0x1F	0789	0x0C	0834	0x0F	0879	0x1D	0924	0x11
0655	0x03	0700	0x16	0745	0x0F	0790	0x06	0835	0x17	0880	0x1E	0925	0x18
0656	0x11	0701	0x0B	0746	0x17	0791	0x13	0836	0x1B	0881	0x0F	0926	0x0C
0657	0x18	0702	0x15	0747	0x0B	0792	0x09	0837	0x1D	0882	0x07	0927	0x16
0658	0x0C	0703	0x1A	0748	0x05	0793	0x14	0838	0x0E	0883	0x03	0928	0x1B
0659	0x06	0704	0x0D	0749	0x12	0794	0x0A	0839	0x17	0884	0x11	0929	0x0D
0660	0x03	0705	0x16	0750	0x09	0795	0x15	0840	0x0B	0885	0x08	0930	0x16
0661	0x01	0706	0x1B	0751	0x04	0796	0x1A	0841	0x15	0886	0x04	0931	0x1B
0662	0x00	0707	0x0D	0752	0x02	0797	0x1D	0842	0x1A	0887	0x02	0932	0x0D
0663	0x10	0708	0x16	0753	0x01	0798	0x1E	0843	0x0D	0888	0x01	0933	0x06
0664	0x08	0709	0x1B	0754	0x10	0799	0x0F	0844	0x16	0889	0x00	0934	0x13
0665	0x14	0710	0x0D	0755	0x18	0800	0x17	0845	0x1B	0890	0x10	0935	0x09
0666	0x0A	0711	0x16	0756	0x1C	0801	0x0B	0846	0x1D	0891	0x08	0936	0x04
0667	0x15	0712	0x0B	0757	0x0E	0802	0x05	0847	0x0E	0892	0x04	0937	0x02
0668	0x0A	0713	0x05	0758	0x17	0803	0x12	0848	0x17	0893	0x02	0938	0x01
0669	0x05	0714	0x02	0759	0x0B	0804	0x19	0849	0x0B	0894	0x01	0939	0x10
0670	0x02	0715	0x01	0760	0x05	0805	0x0C	0850	0x15	0895	0x10	0940	0x08
0671	0x01	0716	0x00	0761	0x02	0806	0x16	0851	0x0A	0896	0x08	0941	0x14
0672	0x00	0717	0x00	0762	0x01	0807	0x0B	0852	0x15	0897	0x04	0942	0x1A
0673	0x00	0718	0x10	0763	0x10	0808	0x15	0853	0x1A	0898	0x12	0943	0x1D
0674	0x10	0719	0x08	0764	0x08	0809	0x1A	0854	0x1D	0899	0x09	0944	0x0E

NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value
0945	0x07	0990	0x06	1035	0x07	1080	0x0E	1125	0x0D	1170	0x03	1215	0x11
0946	0x13	0991	0x03	1036	0x13	1081	0x07	1126	0x16	1171	0x11	1216	0x08
0947	0x09	0992	0x01	1037	0x19	1082	0x13	1127	0x1B	1172	0x08	1217	0x14
0948	0x14	0993	0x00	1038	0x1C	1083	0x19	1128	0x1D	1173	0x04	1218	0x0A
0949	0x0A	0994	0x10	1039	0x0E	1084	0x1C	1129	0x1E	1174	0x02	1219	0x15
0950	0x05	0995	0x08	1040	0x17	1085	0x1E	1130	0x1F	1175	0x01	1220	0x0A
0951	0x02	0996	0x04	1041	0x0B	1086	0x0F	1131	0x0F	1176	0x10	1221	0x15
0952	0x01	0997	0x02	1042	0x15	1087	0x17	1132	0x17	1177	0x18	1222	0x0A
0953	0x00	0998	0x11	1043	0x1A	1088	0x0B	1133	0x0B	1178	0x0C	1223	0x15
0954	0x00	0999	0x08	1044	0x0D	1089	0x15	1134	0x15	1179	0x06	1224	0x0A
0955	0x10	1000	0x04	1045	0x06	1090	0x0A	1135	0x0A	1180	0x03	1225	0x05
0956	0x18	1001	0x02	1046	0x03	1091	0x15	1136	0x15	1181	0x01	1226	0x12
0957	0x1C	1002	0x11	1047	0x11	1092	0x1A	1137	0x0A	1182	0x00	1227	0x19
0958	0x1E	1003	0x08	1048	0x08	1093	0x1D	1138	0x05	1183	0x00	1228	0x1C
0959	0x1F	1004	0x14	1049	0x14	1094	0x0E	1139	0x02	1184	0x10	1229	0x1E
0960	0x1F	1005	0x1A	1050	0x0A	1095	0x07	1140	0x01	1185	0x18	1230	0x1F
0961	0x1F	1006	0x0D	1051	0x05	1096	0x03	1141	0x00	1186	0x0C	1231	0x0F
0962	0x1F	1007	0x06	1052	0x12	1097	0x11	1142	0x10	1187	0x16	1232	0x07
0963	0x0F	1008	0x13	1053	0x09	1098	0x08	1143	0x18	1188	0x0B	1233	0x03
0964	0x07	1009	0x19	1054	0x04	1099	0x14	1144	0x0C	1189	0x15	1234	0x11
0965	0x03	1010	0x1C	1055	0x12	1100	0x0A	1145	0x16	1190	0x1A	1235	0x18
0966	0x11	1011	0x1E	1056	0x09	1101	0x05	1146	0x1B	1191	0x1D	1236	0x0C
0967	0x08	1012	0x1F	1057	0x14	1102	0x12	1147	0x1D	1192	0x1E	1237	0x06
0968	0x04	1013	0x0F	1058	0x0A	1103	0x19	1148	0x0E	1193	0x0F	1238	0x13
0969	0x12	1014	0x07	1059	0x15	1104	0x0C	1149	0x17	1194	0x07	1239	0x09
0970	0x19	1015	0x03	1060	0x0A	1105	0x16	1150	0x1B	1195	0x13	1240	0x14
0971	0x0C	1016	0x01	1061	0x05	1106	0x0B	1151	0x0D	1196	0x19	1241	0x1A
0972	0x16	1017	0x10	1062	0x02	1107	0x15	1152	0x16	1197	0x1C	1242	0x0D
0973	0x0B	1018	0x08	1063	0x11	1108	0x0A	1153	0x1B	1198	0x1E	1243	0x16
0974	0x05	1019	0x04	1064	0x18	1109	0x05	1154	0x1D	1199	0x0F	1244	0x1B
0975	0x12	1020	0x12	1065	0x0C	1110	0x02	1155	0x1E	1200	0x07	1245	0x1D
0976	0x19	1021	0x09	1066	0x16	1111	0x11	1156	0x0F	1201	0x13	1246	0x0E
0977	0x0C	1022	0x04	1067	0x0B	1112	0x08	1157	0x17	1202	0x19	1247	0x07
0978	0x16	1023	0x12	1068	0x15	1113	0x14	1158	0x0B	1203	0x0C	1248	0x13
0979	0x0B	1024	0x19	1069	0x0A	1114	0x0A	1159	0x05	1204	0x16	1249	0x19
0980	0x05	1025	0x1C	1070	0x05	1115	0x05	1160	0x12	1205	0x1B	1250	0x0C
0981	0x12	1026	0x1E	1071	0x02	1116	0x12	1161	0x09	1206	0x0D	1251	0x16
0982	0x09	1027	0x0F	1072	0x01	1117	0x09	1162	0x04	1207	0x16	1252	0x1B
0983	0x04	1028	0x07	1073	0x00	1118	0x14	1163	0x12	1208	0x1B	1253	0x0D
0984	0x12	1029	0x03	1074	0x10	1119	0x1A	1164	0x09	1209	0x1D	1254	0x06
0985	0x09	1030	0x11	1075	0x08	1120	0x0D	1165	0x14	1210	0x1E	1255	0x13
0986	0x04	1031	0x18	1076	0x04	1121	0x16	1166	0x1A	1211	0x1F	1256	0x19
0987	0x12	1032	0x1C	1077	0x12	1122	0x0B	1167	0x1D	1212	0x0F	1257	0x1C
0988	0x19	1033	0x1E	1078	0x19	1123	0x15	1168	0x0E	1213	0x07	1258	0x1E
0989	0x0C	1034	0x0F	1079	0x1C	1124	0x1A	1169	0x07	1214	0x03	1259	0x0F

NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value
1260	0x17	1305	0x0F	1350	0x05	1395	0x0A	1440	0x01	1485	0x14	1530	0x17
1261	0x0B	1306	0x07	1351	0x02	1396	0x05	1441	0x00	1486	0x1A	1531	0x0B
1262	0x05	1307	0x13	1352	0x11	1397	0x12	1442	0x00	1487	0x0D	1532	0x05
1263	0x02	1308	0x09	1353	0x08	1398	0x19	1443	0x00	1488	0x16	1533	0x02
1264	0x11	1309	0x04	1354	0x04	1399	0x1C	1444	0x00	1489	0x0B	1534	0x11
1265	0x18	1310	0x02	1355	0x02	1400	0x1E	1445	0x00	1490	0x05	1535	0x08
1266	0x1C	1311	0x01	1356	0x01	1401	0x0F	1446	0x10	1491	0x12	1536	0x04
1267	0x0E	1312	0x00	1357	0x00	1402	0x07	1447	0x18	1492	0x09	1537	0x12
1268	0x17	1313	0x10	1358	0x00	1403	0x03	1448	0x1C	1493	0x14	1538	0x19
1269	0x0B	1314	0x08	1359	0x10	1404	0x01	1449	0x0E	1494	0x1A	1539	0x0C
1270	0x05	1315	0x14	1360	0x08	1405	0x10	1450	0x07	1495	0x1D	1540	0x06
1271	0x02	1316	0x1A	1361	0x04	1406	0x18	1451	0x03	1496	0x0E	1541	0x13
1272	0x11	1317	0x0D	1362	0x02	1407	0x1C	1452	0x11	1497	0x17	1542	0x09
1273	0x18	1318	0x16	1363	0x11	1408	0x0E	1453	0x18	1498	0x1B	1543	0x14
1274	0x0C	1319	0x0B	1364	0x18	1409	0x07	1454	0x0C	1499	0x1D	1544	0x1A
1275	0x16	1320	0x05	1365	0x1C	1410	0x03	1455	0x16	1500	0x0E	1545	0x0D
1276	0x1B	1321	0x02	1366	0x0E	1411	0x11	1456	0x1B	1501	0x07	1546	0x06
1277	0x0D	1322	0x11	1367	0x17	1412	0x08	1457	0x1D	1502	0x03	1547	0x13
1278	0x06	1323	0x18	1368	0x0B	1413	0x04	1458	0x1E	1503	0x11	1548	0x19
1279	0x03	1324	0x1C	1369	0x15	1414	0x12	1459	0x1F	1504	0x08	1549	0x0C
1280	0x01	1325	0x1E	1370	0x0A	1415	0x19	1460	0x0F	1505	0x04	1550	0x06
1281	0x00	1326	0x0F	1371	0x15	1416	0x1C	1461	0x07	1506	0x12	1551	0x13
1282	0x10	1327	0x07	1372	0x1A	1417	0x1E	1462	0x13	1507	0x09	1552	0x09
1283	0x08	1328	0x03	1373	0x0D	1418	0x0F	1463	0x19	1508	0x14	1553	0x04
1284	0x14	1329	0x01	1374	0x16	1419	0x07	1464	0x0C	1509	0x1A	1554	0x02
1285	0x1A	1330	0x10	1375	0x0B	1420	0x13	1465	0x16	1510	0x0D	1555	0x11
1286	0x1D	1331	0x08	1376	0x05	1421	0x19	1466	0x0B	1511	0x06	1556	0x08
1287	0x0E	1332	0x14	1377	0x12	1422	0x1C	1467	0x15	1512	0x13	1557	0x14
1288	0x07	1333	0x0A	1378	0x09	1423	0x1E	1468	0x1A	1513	0x09	1558	0x1A
1289	0x03	1334	0x05	1379	0x04	1424	0x1F	1469	0x1D	1514	0x14	1559	0x1D
1290	0x01	1335	0x02	1380	0x12	1425	0x0F	1470	0x1E	1515	0x0A	1560	0x1E
1291	0x10	1336	0x11	1381	0x19	1426	0x17	1471	0x1F	1516	0x15	1561	0x1F
1292	0x08	1337	0x18	1382	0x0C	1427	0x1B	1472	0x0F	1517	0x0A	1562	0x1F
1293	0x04	1338	0x0C	1383	0x16	1428	0x0D	1473	0x17	1518	0x05	1563	0x1F
1294	0x02	1339	0x06	1384	0x1B	1429	0x16	1474	0x1B	1519	0x12	1564	0x1F
1295	0x11	1340	0x13	1385	0x0D	1430	0x0B	1475	0x1D	1520	0x09	1565	0x0F
1296	0x08	1341	0x19	1386	0x06	1431	0x05	1476	0x1E	1521	0x14	1566	0x17
1297	0x14	1342	0x0C	1387	0x13	1432	0x02	1477	0x1F	1522	0x0A	1567	0x1B
1298	0x1A	1343	0x06	1388	0x09	1433	0x11	1478	0x0F	1523	0x15	1568	0x1D
1299	0x1D	1344	0x03	1389	0x14	1434	0x18	1479	0x07	1524	0x0A	1569	0x0E
1300	0x0E	1345	0x11	1390	0x1A	1435	0x0C	1480	0x13	1525	0x05	1570	0x07
1301	0x17	1346	0x18	1391	0x0D	1436	0x16	1481	0x09	1526	0x12	1571	0x13
1302	0x1B	1347	0x0C	1392	0x16	1437	0x0B	1482	0x04	1527	0x19	1572	0x09
1303	0x1D	1348	0x16	1393	0x0B	1438	0x05	1483	0x12	1528	0x1C	1573	0x14
1304	0x1E	1349	0x0B	1394	0x15	1439	0x02	1484	0x09	1529	0x0E	1574	0x0A

NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value
1575	0x15	1620	0x0E	1665	0x00	1710	0x06	1755	0x14	1800	0x04	1845	0x1F
1576	0x0A	1621	0x07	1666	0x00	1711	0x13	1756	0x0A	1801	0x12	1846	0x1F
1577	0x05	1622	0x13	1667	0x10	1712	0x19	1757	0x15	1802	0x09	1847	0x1F
1578	0x02	1623	0x19	1668	0x18	1713	0x0C	1758	0x1A	1803	0x04	1848	0x1F
1579	0x01	1624	0x0C	1669	0x0C	1714	0x06	1759	0x1D	1804	0x02	1849	0x1F
1580	0x10	1625	0x06	1670	0x16	1715	0x13	1760	0x1E	1805	0x01	1850	0x0F
1581	0x08	1626	0x13	1671	0x1B	1716	0x09	1761	0x1F	1806	0x10	1851	0x17
1582	0x14	1627	0x19	1672	0x0D	1717	0x14	1762	0x1F	1807	0x18	1852	0x1B
1583	0x0A	1628	0x0C	1673	0x16	1718	0x0A	1763	0x0F	1808	0x0C	1853	0x0D
1584	0x15	1629	0x16	1674	0x1B	1719	0x15	1764	0x07	1809	0x06	1854	0x06
1585	0x1A	1630	0x1B	1675	0x1D	1720	0x0A	1765	0x13	1810	0x13	1855	0x03
1586	0x0D	1631	0x0D	1676	0x0E	1721	0x15	1766	0x19	1811	0x09	1856	0x11
1587	0x16	1632	0x16	1677	0x17	1722	0x1A	1767	0x1C	1812	0x04	1857	0x08
1588	0x0B	1633	0x0B	1678	0x0B	1723	0x0D	1768	0x1E	1813	0x02	1858	0x14
1589	0x05	1634	0x15	1679	0x05	1724	0x16	1769	0x1F	1814	0x11	1859	0x1A
1590	0x02	1635	0x1A	1680	0x02	1725	0x0B	1770	0x1F	1815	0x18	1860	0x1D
1591	0x01	1636	0x1D	1681	0x11	1726	0x15	1771	0x1F	1816	0x1C	1861	0x1E
1592	0x00	1637	0x0E	1682	0x18	1727	0x1A	1772	0x1F	1817	0x1E	1862	0x0F
1593	0x10	1638	0x07	1683	0x1C	1728	0x0D	1773	0x0F	1818	0x1F	1863	0x07
1594	0x18	1639	0x13	1684	0x1E	1729	0x06	1774	0x17	1819	0x1F	1864	0x13
1595	0x0C	1640	0x19	1685	0x1F	1730	0x13	1775	0x0B	1820	0x1F	1865	0x19
1596	0x06	1641	0x1C	1686	0x0F	1731	0x19	1776	0x15	1821	0x0F	1866	0x0C
1597	0x13	1642	0x1E	1687	0x07	1732	0x1C	1777	0x0A	1822	0x17	1867	0x16
1598	0x19	1643	0x0F	1688	0x03	1733	0x1E	1778	0x05	1823	0x0B	1868	0x0B
1599	0x0C	1644	0x07	1689	0x11	1734	0x0F	1779	0x12	1824	0x05	1869	0x05
1600	0x16	1645	0x03	1690	0x08	1735	0x07	1780	0x09	1825	0x12	1870	0x12
1601	0x1B	1646	0x11	1691	0x04	1736	0x03	1781	0x04	1826	0x09	1871	0x19
1602	0x1D	1647	0x08	1692	0x02	1737	0x01	1782	0x02	1827	0x14	1872	0x1C
1603	0x1E	1648	0x14	1693	0x11	1738	0x00	1783	0x01	1828	0x0A	1873	0x1E
1604	0x0F	1649	0x1A	1694	0x08	1739	0x10	1784	0x00	1829	0x05	1874	0x0F
1605	0x17	1650	0x0D	1695	0x14	1740	0x18	1785	0x00	1830	0x12	1875	0x07
1606	0x1B	1651	0x06	1696	0x0A	1741	0x0C	1786	0x10	1831	0x19	1876	0x13
1607	0x1D	1652	0x13	1697	0x05	1742	0x06	1787	0x08	1832	0x1C	1877	0x09
1608	0x1E	1653	0x09	1698	0x02	1743	0x03	1788	0x14	1833	0x1E	1878	0x14
1609	0x0F	1654	0x14	1699	0x11	1744	0x01	1789	0x0A	1834	0x1F	1879	0x1A
1610	0x07	1655	0x1A	1700	0x18	1745	0x00	1790	0x15	1835	0x1F	1880	0x1D
1611	0x13	1656	0x1D	1701	0x1C	1746	0x10	1791	0x1A	1836	0x0F	1881	0x0E
1612	0x19	1657	0x0E	1702	0x0E	1747	0x18	1792	0x1D	1837	0x07	1882	0x17
1613	0x0C	1658	0x07	1703	0x17	1748	0x1C	1793	0x0E	1838	0x13	1883	0x0B
1614	0x06	1659	0x13	1704	0x1B	1749	0x0E	1794	0x07	1839	0x19	1884	0x05
1615	0x03	1660	0x09	1705	0x0D	1750	0x17	1795	0x03	1840	0x0C	1885	0x02
1616	0x01	1661	0x04	1706	0x06	1751	0x0B	1796	0x01	1841	0x16	1886	0x01
1617	0x10	1662	0x02	1707	0x13	1752	0x05	1797	0x00	1842	0x1B	1887	0x00
1618	0x18	1663	0x01	1708	0x19	1753	0x12	1798	0x10	1843	0x1D	1888	0x10
1619	0x1C	1664	0x00	1709	0x0C	1754	0x09	1799	0x08	1844	0x1E	1889	0x18

NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value
1890	0x1C	1935	0x1F	1980	0x08	2025	0x0B	2070	0x00	2115	0x1A	2160	0x0C
1891	0x0E	1936	0x1F	1981	0x04	2026	0x05	2071	0x10	2116	0x0D	2161	0x06
1892	0x17	1937	0x0F	1982	0x02	2027	0x12	2072	0x08	2117	0x06	2162	0x03
1893	0x1B	1938	0x07	1983	0x01	2028	0x19	2073	0x14	2118	0x03	2163	0x01
1894	0x0D	1939	0x03	1984	0x10	2029	0x1C	2074	0x1A	2119	0x01	2164	0x10
1895	0x16	1940	0x01	1985	0x18	2030	0x1E	2075	0x1D	2120	0x10	2165	0x18
1896	0x0B	1941	0x00	1986	0x1C	2031	0x1F	2076	0x0E	2121	0x18	2166	0x1C
1897	0x15	1942	0x00	1987	0x1E	2032	0x0F	2077	0x17	2122	0x0C	2167	0x1E
1898	0x0A	1943	0x10	1988	0x0F	2033	0x17	2078	0x0B	2123	0x06	2168	0x0F
1899	0x05	1944	0x18	1989	0x07	2034	0x0B	2079	0x05	2124	0x13	2169	0x17
1900	0x12	1945	0x0C	1990	0x03	2035	0x15	2080	0x12	2125	0x19	2170	0x1B
1901	0x09	1946	0x06	1991	0x01	2036	0x1A	2081	0x09	2126	0x1C	2171	0x0D
1902	0x14	1947	0x03	1992	0x00	2037	0x0D	2082	0x04	2127	0x0E	2172	0x06
1903	0x1A	1948	0x11	1993	0x00	2038	0x06	2083	0x12	2128	0x17	2173	0x03
1904	0x0D	1949	0x18	1994	0x10	2039	0x03	2084	0x19	2129	0x1B	2174	0x11
1905	0x06	1950	0x1C	1995	0x08	2040	0x01	2085	0x1C	2130	0x1D	2175	0x18
1906	0x03	1951	0x1E	1996	0x04	2041	0x00	2086	0x1E	2131	0x0E	2176	0x1C
1907	0x11	1952	0x1F	1997	0x02	2042	0x10	2087	0x1F	2132	0x07	2177	0x1E
1908	0x18	1953	0x0F	1998	0x01	2043	0x08	2088	0x0F	2133	0x03	2178	0x1F
1909	0x0C	1954	0x07	1999	0x10	2044	0x04	2089	0x07	2134	0x01	2179	0x1F
1910	0x16	1955	0x13	2000	0x18	2045	0x12	2090	0x13	2135	0x00	2180	0x0F
1911	0x0B	1956	0x19	2001	0x0C	2046	0x19	2091	0x19	2136	0x00	2181	0x17
1912	0x15	1957	0x0C	2002	0x16	2047	0x0C	2092	0x1C	2137	0x10	2182	0x1B
1913	0x0A	1958	0x06	2003	0x0B	2048	0x06	2093	0x0E	2138	0x08	2183	0x0D
1914	0x15	1959	0x03	2004	0x05	2049	0x03	2094	0x07	2139	0x14	2184	0x06
1915	0x0A	1960	0x11	2005	0x02	2050	0x11	2095	0x13	2140	0x0A	2185	0x13
1916	0x05	1961	0x18	2006	0x01	2051	0x18	2096	0x19	2141	0x05	2186	0x19
1917	0x02	1962	0x1C	2007	0x10	2052	0x1C	2097	0x1C	2142	0x12	2187	0x0C
1918	0x11	1963	0x1E	2008	0x08	2053	0x0E	2098	0x0E	2143	0x19	2188	0x16
1919	0x08	1964	0x0F	2009	0x04	2054	0x07	2099	0x07	2144	0x0C	2189	0x1B
1920	0x14	1965	0x07	2010	0x02	2055	0x03	2100	0x13	2145	0x06	2190	0x1D
1921	0x1A	1966	0x03	2011	0x01	2056	0x01	2101	0x09	2146	0x03	2191	0x0E
1922	0x0D	1967	0x11	2012	0x10	2057	0x10	2102	0x14	2147	0x11	2192	0x07
1923	0x16	1968	0x18	2013	0x08	2058	0x18	2103	0x0A	2148	0x08	2193	0x13
1924	0x0B	1969	0x0C	2014	0x14	2059	0x1C	2104	0x05	2149	0x04	2194	0x19
1925	0x15	1970	0x16	2015	0x1A	2060	0x1E	2105	0x12	2150	0x02	2195	0x1C
1926	0x1A	1971	0x0B	2016	0x0D	2061	0x1F	2106	0x09	2151	0x01	2196	0x1E
1927	0x1D	1972	0x05	2017	0x06	2062	0x0F	2107	0x04	2152	0x00	2197	0x1F
1928	0x1E	1973	0x12	2018	0x03	2063	0x17	2108	0x02	2153	0x00	2198	0x0F
1929	0x1F	1974	0x19	2019	0x11	2064	0x0B	2109	0x11	2154	0x00	2199	0x07
1930	0x1F	1975	0x0C	2020	0x08	2065	0x05	2110	0x18	2155	0x00	2200	0x13
1931	0x1F	1976	0x06	2021	0x14	2066	0x02	2111	0x0C	2156	0x00	2201	0x09
1932	0x1F	1977	0x03	2022	0x1A	2067	0x01	2112	0x16	2157	0x00	2202	0x14
1933	0x1F	1978	0x01	2023	0x0D	2068	0x00	2113	0x0B	2158	0x10	2203	0x0A
1934	0x1F	1979	0x10	2024	0x16	2069	0x00	2114	0x15	2159	0x18	2204	0x05

NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value
2205	0x12	2250	0x0E	2295	0x12	2340	0x07	2385	0x13	2430	0x1F	2475	0x0B
2206	0x19	2251	0x07	2296	0x09	2341	0x13	2386	0x19	2431	0x1F	2476	0x15
2207	0x0C	2252	0x03	2297	0x14	2342	0x19	2387	0x0C	2432	0x1F	2477	0x0A
2208	0x06	2253	0x11	2298	0x0A	2343	0x1C	2388	0x06	2433	0x1F	2478	0x15
2209	0x13	2254	0x08	2299	0x05	2344	0x1E	2389	0x13	2434	0x0F	2479	0x1A
2210	0x19	2255	0x04	2300	0x12	2345	0x1F	2390	0x09	2435	0x07	2480	0x0D
2211	0x0C	2256	0x02	2301	0x19	2346	0x1F	2391	0x14	2436	0x13	2481	0x16
2212	0x06	2257	0x11	2302	0x0C	2347	0x0F	2392	0x1A	2437	0x09	2482	0x1B
2213	0x03	2258	0x18	2303	0x16	2348	0x17	2393	0x1D	2438	0x04	2483	0x0D
2214	0x01	2259	0x1C	2304	0x1B	2349	0x0B	2394	0x0E	2439	0x02	2484	0x16
2215	0x10	2260	0x0E	2305	0x1D	2350	0x15	2395	0x17	2440	0x11	2485	0x0B
2216	0x08	2261	0x07	2306	0x0E	2351	0x0A	2396	0x1B	2441	0x18	2486	0x05
2217	0x14	2262	0x03	2307	0x07	2352	0x15	2397	0x0D	2442	0x1C	2487	0x12
2218	0x0A	2263	0x11	2308	0x03	2353	0x1A	2398	0x06	2443	0x0E	2488	0x09
2219	0x05	2264	0x08	2309	0x11	2354	0x0D	2399	0x03	2444	0x17	2489	0x04
2220	0x12	2265	0x14	2310	0x18	2355	0x06	2400	0x01	2445	0x1B	2490	0x02
2221	0x19	2266	0x1A	2311	0x1C	2356	0x03	2401	0x10	2446	0x1D	2491	0x11
2222	0x1C	2267	0x1D	2312	0x1E	2357	0x01	2402	0x08	2447	0x0E	2492	0x08
2223	0x0E	2268	0x1E	2313	0x0F	2358	0x10	2403	0x14	2448	0x17	2493	0x14
2224	0x07	2269	0x1F	2314	0x17	2359	0x08	2404	0x1A	2449	0x1B	2494	0x1A
2225	0x13	2270	0x0F	2315	0x1B	2360	0x04	2405	0x1D	2450	0x0D	2495	0x0D
2226	0x09	2271	0x17	2316	0x1D	2361	0x02	2406	0x1E	2451	0x06	2496	0x16
2227	0x04	2272	0x1B	2317	0x1E	2362	0x11	2407	0x1F	2452	0x03	2497	0x1B
2228	0x12	2273	0x0D	2318	0x0F	2363	0x18	2408	0x1F	2453	0x11	2498	0x1D
2229	0x09	2274	0x16	2319	0x17	2364	0x1C	2409	0x0F	2454	0x18	2499	0x1E
2230	0x14	2275	0x1B	2320	0x1B	2365	0x1E	2410	0x17	2455	0x0C	2500	0x1F
2231	0x0A	2276	0x0D	2321	0x1D	2366	0x1F	2411	0x0B	2456	0x16	2501	0x1F
2232	0x05	2277	0x06	2322	0x0E	2367	0x0F	2412	0x15	2457	0x1B	2502	0x1F
2233	0x12	2278	0x13	2323	0x07	2368	0x17	2413	0x1A	2458	0x1D	2503	0x0F
2234	0x09	2279	0x19	2324	0x03	2369	0x0B	2414	0x1D	2459	0x0E	2504	0x17
2235	0x04	2280	0x0C	2325	0x11	2370	0x15	2415	0x1E	2460	0x17	2505	0x0B
2236	0x12	2281	0x06	2326	0x18	2371	0x0A	2416	0x0F	2461	0x0B	2506	0x15
2237	0x19	2282	0x03	2327	0x0C	2372	0x05	2417	0x07	2462	0x05	2507	0x1A
2238	0x1C	2283	0x11	2328	0x16	2373	0x02	2418	0x03	2463	0x12	2508	0x0D
2239	0x0E	2284	0x08	2329	0x0B	2374	0x01	2419	0x11	2464	0x19	2509	0x16
2240	0x17	2285	0x04	2330	0x15	2375	0x00	2420	0x18	2465	0x1C	2510	0x0B
2241	0x0B	2286	0x12	2331	0x1A	2376	0x00	2421	0x1C	2466	0x1E	2511	0x05
2242	0x05	2287	0x09	2332	0x1D	2377	0x00	2422	0x0E	2467	0x1F	2512	0x02
2243	0x12	2288	0x04	2333	0x0E	2378	0x00	2423	0x07	2468	0x1F	2513	0x11
2244	0x19	2289	0x02	2334	0x07	2379	0x10	2424	0x03	2469	0x1F	2514	0x08
2245	0x0C	2290	0x01	2335	0x03	2380	0x18	2425	0x11	2470	0x0F	2515	0x14
2246	0x06	2291	0x00	2336	0x11	2381	0x1C	2426	0x18	2471	0x17	2516	0x1A
2247	0x13	2292	0x10	2337	0x18	2382	0x1E	2427	0x1C	2472	0x1B	2517	0x0D
2248	0x19	2293	0x08	2338	0x1C	2383	0x0F	2428	0x1E	2473	0x0D	2518	0x06
2249	0x1C	2294	0x04	2339	0x0E	2384	0x07	2429	0x1F	2474	0x16	2519	0x03

NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value
2520	0x11	2565	0x07	2610	0x1F	2655	0x10	2700	0x10	2745	0x13	2790	0x09
2521	0x18	2566	0x03	2611	0x0F	2656	0x18	2701	0x18	2746	0x09	2791	0x14
2522	0x1C	2567	0x01	2612	0x07	2657	0x1C	2702	0x0C	2747	0x04	2792	0x0A
2523	0x1E	2568	0x10	2613	0x13	2658	0x0E	2703	0x16	2748	0x02	2793	0x05
2524	0x0F	2569	0x08	2614	0x09	2659	0x17	2704	0x1B	2749	0x11	2794	0x02
2525	0x17	2570	0x04	2615	0x14	2660	0x0B	2705	0x0D	2750	0x18	2795	0x11
2526	0x0B	2571	0x02	2616	0x0A	2661	0x15	2706	0x06	2751	0x0C	2796	0x18
2527	0x15	2572	0x01	2617	0x15	2662	0x1A	2707	0x13	2752	0x16	2797	0x0C
2528	0x1A	2573	0x00	2618	0x1A	2663	0x0D	2708	0x19	2753	0x1B	2798	0x16
2529	0x0D	2574	0x10	2619	0x1D	2664	0x16	2709	0x0C	2754	0x1D	2799	0x1B
2530	0x16	2575	0x18	2620	0x0E	2665	0x0B	2710	0x16	2755	0x1E	2800	0x1D
2531	0x1B	2576	0x1C	2621	0x07	2666	0x15	2711	0x0B	2756	0x0F	2801	0x0E
2532	0x0D	2577	0x0E	2622	0x13	2667	0x0A	2712	0x15	2757	0x07	2802	0x07
2533	0x06	2578	0x07	2623	0x09	2668	0x15	2713	0x0A	2758	0x03	2803	0x03
2534	0x13	2579	0x13	2624	0x04	2669	0x0A	2714	0x05	2759	0x11	2804	0x01
2535	0x09	2580	0x09	2625	0x12	2670	0x15	2715	0x12	2760	0x18	2805	0x10
2536	0x14	2581	0x14	2626	0x09	2671	0x1A	2716	0x19	2761	0x0C	2806	0x18
2537	0x0A	2582	0x0A	2627	0x04	2672	0x1D	2717	0x0C	2762	0x06	2807	0x1C
2538	0x05	2583	0x15	2628	0x02	2673	0x0E	2718	0x16	2763	0x03	2808	0x0E
2539	0x12	2584	0x1A	2629	0x01	2674	0x17	2719	0x0B	2764	0x01	2809	0x17
2540	0x09	2585	0x0D	2630	0x10	2675	0x1B	2720	0x05	2765	0x10	2810	0x0B
2541	0x14	2586	0x06	2631	0x08	2676	0x1D	2721	0x02	2766	0x18	2811	0x15
2542	0x0A	2587	0x03	2632	0x04	2677	0x0E	2722	0x01	2767	0x0C	2812	0x0A
2543	0x15	2588	0x11	2633	0x02	2678	0x17	2723	0x00	2768	0x16	2813	0x05
2544	0x1A	2589	0x08	2634	0x11	2679	0x0B	2724	0x10	2769	0x0B	2814	0x12
2545	0x0D	2590	0x04	2635	0x08	2680	0x15	2725	0x08	2770	0x15	2815	0x09
2546	0x16	2591	0x02	2636	0x04	2681	0x0A	2726	0x04	2771	0x1A	2816	0x14
2547	0x1B	2592	0x01	2637	0x12	2682	0x05	2727	0x02	2772	0x0D	2817	0x0A
2548	0x1D	2593	0x10	2638	0x19	2683	0x12	2728	0x11	2773	0x16	2818	0x05
2549	0x0E	2594	0x08	2639	0x0C	2684	0x19	2729	0x18	2774	0x0B	2819	0x02
2550	0x07	2595	0x04	2640	0x16	2685	0x1C	2730	0x0C	2775	0x05	2820	0x01
2551	0x03	2596	0x02	2641	0x1B	2686	0x0E	2731	0x06	2776	0x12	2821	0x10
2552	0x11	2597	0x01	2642	0x0D	2687	0x07	2732	0x13	2777	0x19	2822	0x18
2553	0x08	2598	0x00	2643	0x16	2688	0x03	2733	0x09	2778	0x0C	2823	0x1C
2554	0x14	2599	0x00	2644	0x1B	2689	0x01	2734	0x14	2779	0x16	2824	0x1E
2555	0x1A	2600	0x10	2645	0x0D	2690	0x00	2735	0x0A	2780	0x1B	2825	0x1F
2556	0x0D	2601	0x18	2646	0x16	2691	0x10	2736	0x05	2781	0x0D	2826	0x1F
2557	0x16	2602	0x0C	2647	0x1B	2692	0x08	2737	0x12	2782	0x16	2827	0x0F
2558	0x1B	2603	0x06	2648	0x0D	2693	0x14	2738	0x19	2783	0x0B	2828	0x07
2559	0x0D	2604	0x13	2649	0x06	2694	0x1A	2739	0x1C	2784	0x05	2829	0x03
2560	0x16	2605	0x19	2650	0x03	2695	0x0D	2740	0x0E	2785	0x12	2830	0x01
2561	0x1B	2606	0x1C	2651	0x01	2696	0x06	2741	0x17	2786	0x19	2831	0x00
2562	0x1D	2607	0x1E	2652	0x00	2697	0x03	2742	0x1B	2787	0x0C	2832	0x10
2563	0x1E	2608	0x1F	2653	0x00	2698	0x01	2743	0x0D	2788	0x06	2833	0x18
2564	0x0F	2609	0x1F	2654	0x00	2699	0x00	2744	0x06	2789	0x13	2834	0x1C

NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value
2835	0x0E	2880	0x0B	2925	0x17	2970	0x15	3015	0x08	3060	0x13	3105	0x1A
2836	0x07	2881	0x05	2926	0x1B	2971	0x1A	3016	0x14	3061	0x09	3106	0x1D
2837	0x03	2882	0x02	2927	0x1D	2972	0x1D	3017	0x1A	3062	0x04	3107	0x1E
2838	0x01	2883	0x11	2928	0x1E	2973	0x0E	3018	0x1D	3063	0x12	3108	0x0F
2839	0x10	2884	0x18	2929	0x1F	2974	0x17	3019	0x0E	3064	0x09	3109	0x07
2840	0x08	2885	0x0C	2930	0x1F	2975	0x0B	3020	0x17	3065	0x14	3110	0x03
2841	0x14	2886	0x06	2931	0x0F	2976	0x05	3021	0x0B	3066	0x0A	3111	0x01
2842	0x1A	2887	0x03	2932	0x07	2977	0x12	3022	0x15	3067	0x15	3112	0x00
2843	0x1D	2888	0x01	2933	0x03	2978	0x19	3023	0x1A	3068	0x1A	3113	0x10
2844	0x0E	2889	0x00	2934	0x01	2979	0x0C	3024	0x0D	3069	0x0D	3114	0x08
2845	0x17	2890	0x00	2935	0x10	2980	0x16	3025	0x06	3070	0x06	3115	0x04
2846	0x1B	2891	0x00	2936	0x18	2981	0x1B	3026	0x13	3071	0x13	3116	0x02
2847	0x0D	2892	0x10	2937	0x1C	2982	0x1D	3027	0x19	3072	0x19	3117	0x01
2848	0x16	2893	0x08	2938	0x1E	2983	0x1E	3028	0x0C	3073	0x0C	3118	0x00
2849	0x0B	2894	0x14	2939	0x0F	2984	0x0F	3029	0x16	3074	0x06	3119	0x00
2850	0x05	2895	0x0A	2940	0x07	2985	0x07	3030	0x0B	3075	0x03	3120	0x00
2851	0x12	2896	0x15	2941	0x13	2986	0x13	3031	0x05	3076	0x01	3121	0x10
2852	0x09	2897	0x0A	2942	0x09	2987	0x19	3032	0x02	3077	0x00	3122	0x08
2853	0x14	2898	0x15	2943	0x04	2988	0x1C	3033	0x01	3078	0x00	3123	0x04
2854	0x0A	2899	0x0A	2944	0x02	2989	0x0E	3034	0x10	3079	0x10	3124	0x12
2855	0x15	2900	0x05	2945	0x11	2990	0x07	3035	0x18	3080	0x08	3125	0x19
2856	0x0A	2901	0x02	2946	0x08	2991	0x03	3036	0x0C	3081	0x04	3126	0x0C
2857	0x15	2902	0x01	2947	0x14	2992	0x11	3037	0x06	3082	0x12	3127	0x16
2858	0x1A	2903	0x10	2948	0x0A	2993	0x18	3038	0x03	3083	0x09	3128	0x0B
2859	0x1D	2904	0x18	2949	0x15	2994	0x0C	3039	0x11	3084	0x14	3129	0x15
2860	0x1E	2905	0x1C	2950	0x1A	2995	0x06	3040	0x08	3085	0x1A	3130	0x1A
2861	0x0F	2906	0x0E	2951	0x1D	2996	0x13	3041	0x04	3086	0x0D	3131	0x1D
2862	0x17	2907	0x17	2952	0x1E	2997	0x19	3042	0x12	3087	0x16	3132	0x0E
2863	0x1B	2908	0x1B	2953	0x1F	2998	0x1C	3043	0x19	3088	0x1B	3133	0x17
2864	0x0D	2909	0x0D	2954	0x0F	2999	0x1E	3044	0x0C	3089	0x0D	3134	0x0B
2865	0x16	2910	0x06	2955	0x07	3000	0x0F	3045	0x06	3090	0x16	3135	0x15
2866	0x1B	2911	0x03	2956	0x03	3001	0x17	3046	0x03	3091	0x0B	3136	0x1A
2867	0x0D	2912	0x11	2957	0x11	3002	0x1B	3047	0x01	3092	0x15	3137	0x1D
2868	0x16	2913	0x08	2958	0x18	3003	0x0D	3048	0x10	3093	0x1A	3138	0x0E
2869	0x1B	2914	0x04	2959	0x1C	3004	0x06	3049	0x18	3094	0x0D	3139	0x17
2870	0x1D	2915	0x12	2960	0x0E	3005	0x13	3050	0x0C	3095	0x06	3140	0x1B
2871	0x0E	2916	0x19	2961	0x17	3006	0x19	3051	0x06	3096	0x03	3141	0x0D
2872	0x07	2917	0x1C	2962	0x0B	3007	0x1C	3052	0x03	3097	0x11	3142	0x16
2873	0x03	2918	0x0E	2963	0x15	3008	0x1E	3053	0x11	3098	0x18	3143	0x1B
2874	0x01	2919	0x07	2964	0x1A	3009	0x1F	3054	0x18	3099	0x1C	3144	0x0D
2875	0x00	2920	0x03	2965	0x1D	3010	0x1F	3055	0x0C	3100	0x0E	3145	0x16
2876	0x10	2921	0x11	2966	0x1E	3011	0x0F	3056	0x16	3101	0x07	3146	0x0B
2877	0x18	2922	0x18	2967	0x0F	3012	0x07	3057	0x1B	3102	0x13	3147	0x15
2878	0x0C	2923	0x1C	2968	0x17	3013	0x03	3058	0x0D	3103	0x09	3148	0x0A
2879	0x16	2924	0x0E	2969	0x0B	3014	0x11	3059	0x06	3104	0x14	3149	0x05

NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value
3150	0x02	3195	0x0A	3240	0x14	3285	0x0D	3330	0x19	3375	0x1C	3420	0x0E
3151	0x01	3196	0x05	3241	0x0A	3286	0x06	3331	0x0C	3376	0x1E	3421	0x07
3152	0x10	3197	0x12	3242	0x05	3287	0x03	3332	0x06	3377	0x1F	3422	0x13
3153	0x18	3198	0x19	3243	0x02	3288	0x01	3333	0x03	3378	0x1F	3423	0x09
3154	0x0C	3199	0x1C	3244	0x11	3289	0x10	3334	0x11	3379	0x1F	3424	0x04
3155	0x06	3200	0x1E	3245	0x18	3290	0x18	3335	0x08	3380	0x1F	3425	0x12
3156	0x13	3201	0x0F	3246	0x1C	3291	0x1C	3336	0x14	3381	0x1F	3426	0x19
3157	0x19	3202	0x17	3247	0x1E	3292	0x1E	3337	0x0A	3382	0x1F	3427	0x1C
3158	0x0C	3203	0x0B	3248	0x1F	3293	0x1F	3338	0x05	3383	0x0F	3428	0x0E
3159	0x06	3204	0x05	3249	0x1F	3294	0x1F	3339	0x02	3384	0x17	3429	0x07
3160	0x13	3205	0x12	3250	0x0F	3295	0x1F	3340	0x01	3385	0x0B	3430	0x13
3161	0x19	3206	0x19	3251	0x07	3296	0x0F	3341	0x00	3386	0x05	3431	0x09
3162	0x1C	3207	0x1C	3252	0x13	3297	0x07	3342	0x10	3387	0x02	3432	0x14
3163	0x0E	3208	0x1E	3253	0x09	3298	0x03	3343	0x08	3388	0x01	3433	0x1A
3164	0x17	3209	0x0F	3254	0x04	3299	0x01	3344	0x14	3389	0x10	3434	0x0D
3165	0x0B	3210	0x17	3255	0x12	3300	0x10	3345	0x1A	3390	0x08	3435	0x16
3166	0x15	3211	0x1B	3256	0x19	3301	0x08	3346	0x1D	3391	0x04	3436	0x0B
3167	0x0A	3212	0x0D	3257	0x1C	3302	0x14	3347	0x1E	3392	0x12	3437	0x05
3168	0x05	3213	0x16	3258	0x1E	3303	0x1A	3348	0x0F	3393	0x09	3438	0x02
3169	0x02	3214	0x1B	3259	0x0F	3304	0x0D	3349	0x07	3394	0x14	3439	0x01
3170	0x01	3215	0x1D	3260	0x17	3305	0x06	3350	0x03	3395	0x0A	3440	0x10
3171	0x10	3216	0x1E	3261	0x0B	3306	0x13	3351	0x11	3396	0x15	3441	0x18
3172	0x08	3217	0x1F	3262	0x15	3307	0x19	3352	0x08	3397	0x1A	3442	0x1C
3173	0x04	3218	0x1F	3263	0x1A	3308	0x0C	3353	0x14	3398	0x1D	3443	0x0E
3174	0x02	3219	0x0F	3264	0x1D	3309	0x16	3354	0x0A	3399	0x0E	3444	0x07
3175	0x11	3220	0x07	3265	0x1E	3310	0x1B	3355	0x05	3400	0x17	3445	0x13
3176	0x18	3221	0x13	3266	0x1F	3311	0x0D	3356	0x02	3401	0x1B	3446	0x09
3177	0x0C	3222	0x09	3267	0x0F	3312	0x06	3357	0x11	3402	0x0D	3447	0x04
3178	0x16	3223	0x14	3268	0x07	3313	0x03	3358	0x08	3403	0x06	3448	0x02
3179	0x1B	3224	0x1A	3269	0x13	3314	0x11	3359	0x14	3404	0x13	3449	0x11
3180	0x0D	3225	0x1D	3270	0x19	3315	0x18	3360	0x0A	3405	0x09	3450	0x18
3181	0x16	3226	0x1E	3271	0x1C	3316	0x1C	3361	0x15	3406	0x14	3451	0x0C
3182	0x0B	3227	0x1F	3272	0x1E	3317	0x0E	3362	0x1A	3407	0x0A	3452	0x06
3183	0x05	3228	0x0F	3273	0x0F	3318	0x17	3363	0x0D	3408	0x15	3453	0x13
3184	0x02	3229	0x07	3274	0x17	3319	0x1B	3364	0x16	3409	0x1A	3454	0x19
3185	0x11	3230	0x03	3275	0x1B	3320	0x0D	3365	0x1B	3410	0x0D	3455	0x1C
3186	0x08	3231	0x01	3276	0x1D	3321	0x16	3366	0x0D	3411	0x16	3456	0x0E
3187	0x04	3232	0x10	3277	0x0E	3322	0x1B	3367	0x06	3412	0x0B	3457	0x07
3188	0x12	3233	0x18	3278	0x17	3323	0x1D	3368	0x03	3413	0x15	3458	0x13
3189	0x09	3234	0x0C	3279	0x1B	3324	0x0E	3369	0x01	3414	0x0A	3459	0x19
3190	0x04	3235	0x16	3280	0x1D	3325	0x07	3370	0x10	3415	0x05	3460	0x0C
3191	0x02	3236	0x0B	3281	0x1E	3326	0x13	3371	0x08	3416	0x02	3461	0x06
3192	0x11	3237	0x05	3282	0x0F	3327	0x09	3372	0x04	3417	0x11	3462	0x03
3193	0x08	3238	0x12	3283	0x17	3328	0x04	3373	0x12	3418	0x18	3463	0x11
3194	0x14	3239	0x09	3284	0x1B	3329	0x12	3374	0x19	3419	0x1C	3464	0x08

NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value
3465	0x14	3510	0x13	3555	0x17	3600	0x00	3645	0x17	3690	0x0E	3735	0x0F
3466	0x1A	3511	0x19	3556	0x0B	3601	0x00	3646	0x1B	3691	0x07	3736	0x07
3467	0x0D	3512	0x0C	3557	0x05	3602	0x00	3647	0x1D	3692	0x13	3737	0x03
3468	0x06	3513	0x06	3558	0x02	3603	0x00	3648	0x0E	3693	0x19	3738	0x11
3469	0x03	3514	0x03	3559	0x11	3604	0x10	3649	0x17	3694	0x0C	3739	0x18
3470	0x01	3515	0x01	3560	0x08	3605	0x08	3650	0x1B	3695	0x16	3740	0x1C
3471	0x10	3516	0x00	3561	0x04	3606	0x04	3651	0x1D	3696	0x0B	3741	0x1E
3472	0x18	3517	0x10	3562	0x02	3607	0x12	3652	0x0E	3697	0x15	3742	0x1F
3473	0x1C	3518	0x18	3563	0x11	3608	0x09	3653	0x07	3698	0x0A	3743	0x0F
3474	0x0E	3519	0x0C	3564	0x08	3609	0x04	3654	0x13	3699	0x15	3744	0x17
3475	0x17	3520	0x06	3565	0x04	3610	0x12	3655	0x19	3700	0x1A	3745	0x1B
3476	0x1B	3521	0x13	3566	0x12	3611	0x09	3656	0x0C	3701	0x1D	3746	0x1D
3477	0x1D	3522	0x09	3567	0x09	3612	0x14	3657	0x06	3702	0x0E	3747	0x0E
3478	0x0E	3523	0x04	3568	0x04	3613	0x1A	3658	0x13	3703	0x17	3748	0x07
3479	0x07	3524	0x12	3569	0x12	3614	0x0D	3659	0x09	3704	0x0B	3749	0x03
3480	0x13	3525	0x19	3570	0x19	3615	0x06	3660	0x04	3705	0x15	3750	0x01
3481	0x09	3526	0x1C	3571	0x0C	3616	0x03	3661	0x12	3706	0x0A	3751	0x10
3482	0x04	3527	0x1E	3572	0x16	3617	0x01	3662	0x19	3707	0x15	3752	0x08
3483	0x02	3528	0x1F	3573	0x0B	3618	0x10	3663	0x0C	3708	0x0A	3753	0x14
3484	0x11	3529	0x1F	3574	0x05	3619	0x08	3664	0x16	3709	0x15	3754	0x0A
3485	0x08	3530	0x0F	3575	0x02	3620	0x14	3665	0x1B	3710	0x1A	3755	0x15
3486	0x04	3531	0x17	3576	0x11	3621	0x0A	3666	0x1D	3711	0x0D	3756	0x0A
3487	0x02	3532	0x1B	3577	0x08	3622	0x15	3667	0x0E	3712	0x06	3757	0x15
3488	0x11	3533	0x1D	3578	0x14	3623	0x1A	3668	0x17	3713	0x13	3758	0x1A
3489	0x18	3534	0x0E	3579	0x0A	3624	0x1D	3669	0x0B	3714	0x19	3759	0x0D
3490	0x1C	3535	0x17	3580	0x05	3625	0x1E	3670	0x15	3715	0x1C	3760	0x06
3491	0x1E	3536	0x1B	3581	0x02	3626	0x0F	3671	0x1A	3716	0x0E	3761	0x03
3492	0x0F	3537	0x0D	3582	0x01	3627	0x07	3672	0x1D	3717	0x07	3762	0x11
3493	0x07	3538	0x16	3583	0x10	3628	0x03	3673	0x1E	3718	0x03	3763	0x18
3494	0x13	3539	0x0B	3584	0x18	3629	0x01	3674	0x1F	3719	0x01	3764	0x0C
3495	0x09	3540	0x15	3585	0x0C	3630	0x10	3675	0x1F	3720	0x00	3765	0x06
3496	0x14	3541	0x1A	3586	0x16	3631	0x18	3676	0x0F	3721	0x00	3766	0x03
3497	0x0A	3542	0x0D	3587	0x1B	3632	0x0C	3677	0x17	3722	0x00	3767	0x11
3498	0x15	3543	0x16	3588	0x1D	3633	0x06	3678	0x1B	3723	0x10	3768	0x08
3499	0x0A	3544	0x0B	3589	0x0E	3634	0x03	3679	0x1D	3724	0x18	3769	0x14
3500	0x15	3545	0x15	3590	0x07	3635	0x01	3680	0x1E	3725	0x0C	3770	0x0A
3501	0x0A	3546	0x1A	3591	0x13	3636	0x10	3681	0x1F	3726	0x06	3771	0x15
3502	0x05	3547	0x1D	3592	0x09	3637	0x08	3682	0x1F	3727	0x13	3772	0x0A
3503	0x12	3548	0x0E	3593	0x14	3638	0x14	3683	0x0F	3728	0x09	3773	0x05
3504	0x09	3549	0x17	3594	0x1A	3639	0x1A	3684	0x17	3729	0x14	3774	0x02
3505	0x14	3550	0x1B	3595	0x1D	3640	0x0D	3685	0x0B	3730	0x1A	3775	0x11
3506	0x1A	3551	0x1D	3596	0x0E	3641	0x16	3686	0x05	3731	0x1D	3776	0x08
3507	0x1D	3552	0x1E	3597	0x07	3642	0x1B	3687	0x12	3732	0x1E	3777	0x04
3508	0x0E	3553	0x1F	3598	0x03	3643	0x1D	3688	0x19	3733	0x1F	3778	0x12
3509	0x07	3554	0x0F	3599	0x01	3644	0x0E	3689	0x1C	3734	0x1F	3779	0x09

NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value	NO	/ Value
3780	0x14	3825	0x10	3870	0x0A	3915	0x1F	3960	0x1B	4005	0x00	4050	0x19
3781	0x0A	3826	0x08	3871	0x15	3916	0x0F	3961	0x0D	4006	0x00	4051	0x0C
3782	0x15	3827	0x14	3872	0x0A	3917	0x07	3962	0x06	4007	0x00	4052	0x16
3783	0x0A	3828	0x1A	3873	0x15	3918	0x03	3963	0x03	4008	0x10	4053	0x1B
3784	0x15	3829	0x0D	3874	0x1A	3919	0x01	3964	0x11	4009	0x18	4054	0x1D
3785	0x0A	3830	0x16	3875	0x1D	3920	0x00	3965	0x08	4010	0x0C	4055	0x1E
3786	0x15	3831	0x1B	3876	0x1E	3921	0x10	3966	0x14	4011	0x16	4056	0x1F
3787	0x1A	3832	0x0D	3877	0x1F	3922	0x08	3967	0x0A	4012	0x0B	4057	0x0F
3788	0x0D	3833	0x06	3878	0x1F	3923	0x14	3968	0x15	4013	0x05	4058	0x17
3789	0x16	3834	0x13	3879	0x1F	3924	0x0A	3969	0x1A	4014	0x12	4059	0x1B
3790	0x1B	3835	0x19	3880	0x0F	3925	0x05	3970	0x0D	4015	0x19	4060	0x1D
3791	0x1D	3836	0x1C	3881	0x17	3926	0x02	3971	0x06	4016	0x1C	4061	0x0E
3792	0x1E	3837	0x0E	3882	0x1B	3927	0x01	3972	0x13	4017	0x0E	4062	0x17
3793	0x0F	3838	0x07	3883	0x1D	3928	0x00	3973	0x09	4018	0x07	4063	0x0B
3794	0x07	3839	0x13	3884	0x1E	3929	0x00	3974	0x04	4019	0x03	4064	0x05
3795	0x13	3840	0x09	3885	0x0F	3930	0x00	3975	0x02	4020	0x11	4065	0x12
3796	0x09	3841	0x04	3886	0x17	3931	0x10	3976	0x01	4021	0x08	4066	0x09
3797	0x14	3842	0x02	3887	0x0B	3932	0x18	3977	0x00	4022	0x14	4067	0x14
3798	0x0A	3843	0x01	3888	0x15	3933	0x1C	3978	0x00	4023	0x0A	4068	0x1A
3799	0x05	3844	0x10	3889	0x0A	3934	0x1E	3979	0x10	4024	0x15	4069	0x1D
3800	0x02	3845	0x08	3890	0x05	3935	0x1F	3980	0x18	4025	0x1A	4070	0x1E
3801	0x11	3846	0x04	3891	0x02	3936	0x0F	3981	0x1C	4026	0x1D	4071	0x1F
3802	0x08	3847	0x12	3892	0x11	3937	0x17	3982	0x0E	4027	0x0E	4072	0x1F
3803	0x04	3848	0x19	3893	0x08	3938	0x1B	3983	0x17	4028	0x17	4073	0x1F
3804	0x12	3849	0x0C	3894	0x04	3939	0x0D	3984	0x1B	4029	0x0B	4074	0x0F
3805	0x19	3850	0x06	3895	0x02	3940	0x06	3985	0x1D	4030	0x05	4075	0x07
3806	0x1C	3851	0x13	3896	0x01	3941	0x03	3986	0x1E	4031	0x02	4076	0x13
3807	0x0E	3852	0x19	3897	0x10	3942	0x01	3987	0x0F	4032	0x11	4077	0x19
3808	0x17	3853	0x1C	3898	0x08	3943	0x00	3988	0x17	4033	0x08	4078	0x1C
3809	0x0B	3854	0x1E	3899	0x14	3944	0x10	3989	0x0B	4034	0x14	4079	0x0E
3810	0x15	3855	0x0F	3900	0x0A	3945	0x18	3990	0x05	4035	0x1A	4080	0x17
3811	0x1A	3856	0x07	3901	0x05	3946	0x1C	3991	0x02	4036	0x1D	4081	0x1B
3812	0x1D	3857	0x13	3902	0x02	3947	0x1E	3992	0x01	4037	0x0E	4082	0x1D
3813	0x0E	3858	0x09	3903	0x01	3948	0x1F	3993	0x00	4038	0x07	4083	0x1E
3814	0x07	3859	0x04	3904	0x10	3949	0x0F	3994	0x10	4039	0x13	4084	0x0F
3815	0x13	3860	0x12	3905	0x08	3950	0x07	3995	0x08	4040	0x19	4085	0x07
3816	0x09	3861	0x19	3906	0x04	3951	0x13	3996	0x14	4041	0x1C	4086	0x03
3817	0x14	3862	0x0C	3907	0x12	3952	0x09	3997	0x0A	4042	0x0E	4087	0x01
3818	0x1A	3863	0x16	3908	0x19	3953	0x04	3998	0x15	4043	0x07	4088	0x00
3819	0x0D	3864	0x0B	3909	0x1C	3954	0x02	3999	0x1A	4044	0x03	4089	0x00
3820	0x06	3865	0x15	3910	0x0E	3955	0x01	4000	0x0D	4045	0x01	4090	0x00
3821	0x03	3866	0x0A	3911	0x17	3956	0x10	4001	0x06	4046	0x10	4091	0x00
3822	0x01	3867	0x15	3912	0x1B	3957	0x18	4002	0x03	4047	0x08	4092	0x00
3823	0x00	3868	0x0A	3913	0x1D	3958	0x0C	4003	0x01	4048	0x04	4093	0x00
3824	0x00	3869	0x15	3914	0x1E	3959	0x16	4004	0x00	4049	0x12	4094	0x00

LIST OF REFERENCES

1. P. E. Pace, D. J. Fouts, S. Ekestorm, and C. Karow, “Digital False-Target Image Synthesizer for countering ISAR,” *IEE Proceedings F – Radar, Sonar and Navigation*, Vol. 149, No. 5, pp. 248-257, June 2002.

2. D. J. Fouts, P. E. Pace, C. Karow, and S. Ekestorm, “A Single-Chip False Target Radar Image Generator for countering Wideband Imaging Radars,” *IEEE Journal on Solid State Circuits*, Vol. 37, No. 6, pp. 751-759, June 2002.

3. C. A. Amundson, “Design, implementation, and testing of a high performance summation adder for radar image synthesis,” Master’s thesis, Naval Postgraduate School, Monterey, California, September 2001.

4. C. H. Guillaume, “Circuit design and simulation for a digital image synthesizer range bin modulator,” Master’s thesis, Naval Postgraduate School, Monterey, California, March 2002.

5. K. M. Kirin, “VLSI design of SINE/COSINE lookup table for use with digital image synthesizer ASIC,” Master’s thesis, Naval Postgraduate School, Monterey, California, June 2001.

6. B. Ozguvenc, “Mask layout of an ASIC for generating false target radar images,” Master’s thesis, Naval Postgraduate School, Monterey, California, March 2002.

7. F. A. Le Dantec, “Performance analysis of a digital image synthesizer as a counter-measure against inverse synthetic aperture radar,” Master’s thesis, Naval Postgraduate School, Monterey, California, September 2002.

8. H. Bergon, “VHDL modeling and simulation for a digital target imaging architecture for multiple large targets generation,” Master’s thesis, Naval Postgraduate School, Monterey, California, September 2002.

9. D. T. Mattox, “Mask design and layout for a 512-Processor ASIC for generating false target radar images,” Master’s thesis, Naval Postgraduate School, Monterey, California, December 2002.

10. R. C. Altmeyer, “Design, implementation and testing of a VLSI high performance ASIC for extracting the phase of a complex signal,” Master’s thesis, Naval Postgraduate School, Monterey, California, September 2002.

11. J. F. Wakerly, *Digital Design Principles & Practices*, 3rd Ed., Prentice Hall, Upper Saddle River, New Jersey, 2001.

12. D. T. Mattox, *Circuit Design, Mask Layout and Verification for a 512 Processor ASIC for Generating False Target Radar Images,* Thesis Presentation, Naval Postgraduate School, Monterey, California, December 2002.

13. S. Yalamanchili, *Introductory VHDL From Simulation to Synthesis*, Prentice Hall, Upper Saddle River, New Jersey, 2001.

14. http://www.aldec.com/ActiveHDL/ Official site of Aldec, visited until May 2003.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Dr. John P. Powers, Code EC
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California

4. Dr. Douglas J. Fouts, Code EC/Fs
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California

5. Dr. Phillip E. Pace, Code EC/Pc
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California

6. Mr. Alfred A. Di Mattesa
   Naval Research Laboratory
   Washington, D.C.

7. Mr. Gregory P. Hrin
   Naval Research Laboratory
   Washington, D.C.

8. Mr. Daniel W. Bay
   Naval Research Laboratory
   Washington, D.C.

9. Dr. Frank Klemm
   Naval Research Laboratory
   Washington, D.C.

10. Mr. Jeff Knowles
    Naval Research Laboratory
    Washington, D.C.

11. Dr. Joseph Lawrence
Office of Naval Research
Arlington, Virginia

12. Mr. Jim Talley
Office of Naval Research
Arlington, Virginia

13. Mr. Greg Tavik
Office of Naval Research, Radar Division
Arlington, Virginia

14. Mr. Mike Monsma
Office of Naval Research, Radar Division
Arlington, Virginia

15. Mr. Robert Surratt
Office of Naval Research, Radar Division
Arlington, Virginia

16. Dr. Peter Craig
Office of Naval Research
Arlington, Virginia

17. Dr. Will Cronyn
Space and Naval Warfare Systems Command
San Diego, California